AFRL-RI-RS-TR-2007-244
Final Technical Report
November 2007

EFFICIENT PATHFINDING IN VERY LARGE DATA SPACES

Cycorp, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2007-244 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

PETER J. ROCCI, Jr.
Work Unit Manager

/s/

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
NOV 2007	Final	August 2006 - September 2007

4. TITLE AND SUBTITLE

EFFICIENT PATHFINDING IN VERY LARGE DATA SPACES

5a. CONTRACT NUMBER FA8750-06-C-0003

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER 92498F

6. AUTHOR(S)

Douglas B. Lenat, Keith Goolsbey , Kevin Knight and Pace Smith

5d. PROJECT NUMBER RAPD

5e. TASK NUMBER 01

5f. WORK UNIT NUMBER 04

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cycorp, Inc.
7718 Wood Hollow
Austin TX 78731-1645

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RIED
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-RI-RS-TR-2007-244

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0376

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This project created a corpus of large test problems relevant to the Intelligence Community (IC). These problems required bringing together only facts and rules from the unclassified OpenCyc knowledge base. Because many current theorem provers are unable to a reason with (or even load) an IC-sized knowledge base, six different ‘levels’ of problems were created, each containing progressively larger theories. Dozens of Automatic Theorem Proving (ATP) researchers are now heavily engaged in attacking more and more of these six new TPTP problem sets. In addition to challenging the theorem-proving community, this project contained a series of experiments to assess and, where possible, improve the efficiency of Cyc’s general inference engine. These experiments identified areas for immediate improvement, and approximately one full factor of 10 speedup was obtained just in the course of carrying them out and analyzing their results.

15. SUBJECT TERMS

Knowledge Bases, Theorem Provers, Inference, Reasoning

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Peter J. Rocci, Jr.
U	U	U	UL	62	**19b. TELEPHONE NUMBER (*Include area code*)** N/A

Table of Contents

List of Figures and Tables

Abstract

The amount of data available to US Intelligence Community (IC) is growing far faster than the community's ability to identify the paths through this data that represent critical patterns of information. And the IC is held responsible – by the media, hence the public, hence Congress – in hindsight for not having connected an "obvious" set of dots, even when those data were needles hidden in acres of haystacks.

There have been a very large number of automated inference efforts over the past two decades which *could* have served as "forcing functions", as drivers to get researchers and developers to tackle this problem of finding paths (of potential interest to analysts) in enormous datasets. But unfortunately each problem, each project, has had its own idiosyncratic features that could be used to advantage – to cut ontological corners, as it were – and therefore what has evolved, during these two decades, is a proliferation of specialized reasoning systems that fill various narrow niches. Some of them achieve lightning-fast performance by drastically restricting the expressiveness of their representation language, or by prohibiting incremental updates or additions to their data base, or by assuming one consistent, global information-space, etc. We say "unfortunately" because the overall problem of efficient IC pathfinding involves dealing with data which are complicated assertions ("Israeli intelligence now believes that HAMAS will not oppose a U.S. demand that at least some of its…"), a continual stream of realtime updates, multiple contexts and multiple belief systems, etc.

So each of these various specialized reasoning systems, while useful for the problem it was designed and built for, is ill-matched to the full IC pathfinding problem. And yet they are just so damn fast at what they *can* handle, compared to general theorem provers! Is there some way we can take advantage of this, some way we can harness them, for the general IC pathfinding task? That is the question we addressed in this project. Our basic hypothesis was that some of these specialized reasoners might be able to at least solve some sub-goals, some sub-problems. If we could characterize which ones were good for which types of subproblems, and delegate to each one the specific subproblems it would be efficient at working on, then a "community" of *n* reasoners might be able to solve a top-level problem more efficiently than any one of them, in fact in less than $1/n^{th}$ time the that any one of them would have taken for that overall problem. In other words, our original motivation was to identify synergies between state-of-the-art machine reasoning systems such as SPASS, VAMPIRE, E, on the one hand, and Cyc's expressiveness and generality, its ability to model and attack complex problems more representative of those facing the IC, on the other hand.

The project originally aimed to bring together dozens of leading players in the machine reasoning community to identify specific types of problems for which different inference techniques were most effective; and to use Cyc, whose architecture already provides support for a community of reasoning agents (see Figure 1), as the integration test bed. There was a nearly-unanimous positive reaction and willingness to participate in such an effort, by the worldwide theorem proving community (26 out of 28 invitations resulted in such acceptances).

However, just as the project was to begin, its budget was reduced by an order of magnitude. Working with our government technical program managers and "champions", we worked to scale it back in a way that allowed it to still address these core issues, albeit via drastically more tightly focused tasks. In essence, the strategy became one of communicating with, exciting, involving, and thereby engaging (and at least partially "harnessing") the worldwide theorem-proving community. I.e., our goal became to get them to focus not just on the third decimal digit of performance of their reasoning systems when applied to a set of "toy" problems, but also to add into their sights some much larger reasoning problems that are on the order of those needed to do IC pathfinding, larger in both the sense of number of axioms (assertions, data points) and in the sense of more sophisticated sorts of axioms involving quantifiers, modals, etc., as we exemplified above.

To that end, we created a corpus of IC-sized test problems, problems which while large (in cardinality and sophistication) required bringing together only facts and rules from the unclassified OpenCyc knowledge base. We began dialogues with the theorem-proving community: going to their conferences, giving talks there, exciting them about this new level of problem, and formatting and sharing with them this new corpus of problems in their own accepted format, namely TPTP (which stands for Thousands of Problems for Theorem Provers; see http://www.tptp.org). Somewhat to the surprise of their creators, many of the leading automatic theorem proving ("ATP") programs were unable to even *load* these problems, due to their size, and others took literally *months* of cpu time to load a single problem. Because many current theorem provers are unable to a reason with (or even load) a Cyc-sized – an IC-sized – knowledge base, six different 'levels' of problems were created, each containing progressively larger theories. Dozens of ATP researchers all over the world are now heavily engaged in attacking more and more of these six new TPTP problem sets.

In addition to challenging the theorem-proving community, this project contained a series of experiments to assess and, where possible, improve the efficiency of Cyc's general inference engine. These experiments identified areas for immediate improvement, and approximately one full factor of 10 speedup was obtained just in the course of carrying them out and analyzing their results. I.e., Cyc now runs about 10 times faster than it did one year ago, because of this project. This reflects just the lowest-hanging fruit of the experiments; the also provided several indications of opportunities where additional inference techniques could, in the future, be cost-effectively brought to bear, to gain possibly *another* order of magnitude speedup. And we even more firmly believe in our original hypothesis about the utility of characterizing and harnessing a "team" of the world's best specialized reasoners; we now believe that there is yet *a third* order of magnitude speedup possible through that activity. We hope that the U.S. is the first to do that, and that the U.S. intelligence community reaps the early benefits from it.

1. Introduction

The amount of data available to the US Intelligence Community (IC) continues to grow at an accelerating – and often alarming – rate, far outpacing the rate of increase in brute force computing power. Even worse, the connections among this growing mass of data, representing the information patterns indicative of potential threats or opportunities, is expanding faster still. If the IC is to retain its ability to find the proverbial needles in this explosively growing data hayfield, there must be dramatic increases in the efficiency with which data is processed and information gleaned from it.

The good news is that the past decade has shown some promising advances in machine reasoning and automated inference. This included the progress along a number of fronts within the theorem proving community. As initially proposed, the Efficient Pathfinding project aimed to work with this community on a focused set of problems with the dual goals of 1) identifying best-of-breed solutions (e.g., theorem provers) for specific classes of reasoning problems and leveraging those from within more general Cyc reasoning, when appropriate, and 2) using Cyc to generate problem sets that would represent "stretch goals" for those existing technologies.

When the budget and scope of the initially proposed project were substantially scaled back, by approximately one factor of 10, the project objectives were modified accordingly. The spirit of the original proposal was maintained as best it could be: to dramatically increase the community's joint ability to successfully attack problems of a complexity comparable to those faced by the IC, by increasing the effectiveness of individual reasoning systems (including, of course, Cyc) and/or by exploring the synergistic use of multiple theorem provers, allowing each to play to its strength. However, the extent of these activities, in particular the latter, was, of necessity, scaled back.

A set of key tasks were identified that would both produce tangible interim results as well as lay the foundation for subsequent (2008+) actions by, and interactions with, the larger ATP (automated theorem proving) community. These tasks fell into two main categories:

- The first set of tasks comprised a set of experiments that evaluated the efficiency and effectiveness of Cyc's reasoning capabilities. These experiments then explored – and quantitatively measured – the impact of (potential) enhancements to Cyc's reasoning processes, for example different meta-reasoning (tactician) and meta-meta-reasoning (strategist) reasoning modules (see Figure 1). In addition to providing substantial improvements in both inference speed and the quality of the resulting answers, these experiments also established an <u>evaluation framework</u> that could be extended and applied to assessing other reasoning engines. Furthermore, some of these experiments examined the effect of meta-reasoning within Cyc, using, for example, machine learning techniques to determine which

rules often co-occur within a proof. This meta-reasoning is the framework for, and the first step toward, enabling cross-reasoner "harnessing": the integration of additional types of reasoning engines (such as neural networks) that may be better suited to various particular subtasks, achieving as a result a speedup that is potentially dramatically more than linear in the number of different reasoners being harnessed together.

- The second set of tasks focused more directly on using Cyc's knowledge base as a means to assess and stretch the current capabilities of a range of theorem provers. The automated theorem proving community has made substantial strides in part because of their ability to settle on a common representation format and a large corpus of shared evaluations, namely the "Thousands of Problems for Theorem Provers" test compendium (TPTP)—see www.**tptp**.org . Prior to this project, there was a chasm between the types and complexity of knowledge captured in the Cyc knowledge base and that being used by the TPTP community. These heavily-TPTP-tuned theorem provers competed with each other and gradually evolved into a state where they are extremely efficient when addressing a certain class of problems – in some cases very small (a few tens of axioms) problems, and in some cases larger but very narrowly constrained problems (e.g., assigning rooms for courses at a university, given constraints on buildings, teachers, etc., where the constraints may be numerous but they are all very similar and are easily stated within a simple constraint language such as SAT). Those are exactly the sort of problems in the TPTP list, at least until this project got underway. But these theorem provers are constrained in numerous ways that prevent them from successfully solving (and, in many cases, even *loading*) the types of problems that Cyc addresses: large numbers of axioms over a very large vocabulary, and ones that are complicated enough to make it cost-effective to state them in full first order logic or higher order logic. Those are exactly the sorts of problems that the Intelligence Community faces continually: massive amounts of data, and complex assertions (e.g., source x overheard person y say that they were concerned that in some rural city of country z next month there will be a meeting between…) Therefore, part of this project set out to bridge this divide, by creating a series of evaluations, in a form directly usable by the TPTP community, reasoning problems to challenge that community in ways that the TPTP problem set hitherto had not done, IC-sized reasoning problems to get them to identify and overcome some of their current limitations. A fully unclassified Cyc-derived TPTP problem set was presented at the Empirically Successful Automated Reasoning in Large Theories workshop (see http://www.cs.miami.edu/~geoff/Conferences/ESARLT/) last month, at the 21st Conference on Automated Deduction (CADE-21, which was held in Bremen, Germany, from July 15-20, 2007) and that presentation was very well received. It was agreed enthusiastically that those large problems will now officially become *part* of the thousands of problems for theorem provers on the www.TPTP.org website, and will thereby harness the ATP community to drive the further development of IC-community-relevant theorem provers.

A guide to the layout of the remainder of this final report: In the following section, Section 2, the experiments and their resulting impacts on Cyc's reasoning (and, therefore, on efficient pathfinding) are described. Next, in Section 3, the generation of the new TPTP problem set is discussed in detail, along with a description of experiments using this problem set – experiments to explore the feasibility of integrating Cyc with other theorem provers. Finally, Section 4 presents a review of the accomplishments of this project in the context of the originally proposed objectives (i.e., pre-down scoping to meeting the scaled-back budget) and offers suggestions for the potential next steps most likely to bring the IC further along the road of successfully tackling its efficient pathfinding challenges.

2. Experiments Enhancing A Single Theorem Prover's Reasoning Capabilities

Three families of experiments were designed to measure the inference capabilities of a large-KB-oriented theorem prover (Cyc) and to explore and assess the impacts of three different approaches to significantly speeding up its reasoning algorithms.

The resulting improvements provided immediate benefit in terms of improving both speed and quality of Cyc's query answering. But much more significantly, and more broadly applicable, the results of these experiments point to several potential areas for cost-effective insertion of other, different (from Cyc and from each other) reasoning technologies (e.g., SPASS, E, VAMPIRE) into a multi-reasoning-system "harness", as was originally envisioned for this project. If and when the project is able to ramp up to -- or restart and tackle -- its original mission, the results of these experiments will enable us (or whoever takes up that mantle) to jump-start that project, so that the time spent this past year on this part of the effort can be leveraged dramatically and directly.

The three approaches examined were:

1) using reinforcement learning to improve tactical decision-making,

2) leveraging prior inference results on similar queries, and

3) clustering knowledge to effectively reduce the size of the search space for a given query.

These three (families of) experiments, and their results, are described the three subsections, Sections 2.1, 2.2, and 2.3.

The basic inference engine in Cyc (and similar inference engines) consists of a large number of Workers, of varying degrees of generality (e.g., a full first order logic theorem prover) and specificity (e.g., a module that keeps commonly-used transitive relations R fully expanded, so if we assert R(a,b), R(b,c), R(c,d), R(e,f), and ask whether R(a,f), that module will be able to answer affirmatively (and provide the three-step proof) immediately. This typically requires having specialized data structures, and "demons" to update them. Each Worker has some characterization of what sorts of problems it is able to handle, and at what cost in terms of time to first answer, total time, memory cost, chance of returning at least one answer (if there is one), chance of returning all answers, etc. At a meta-level, there are some modules called Tacticians that decide which Worker to call on, how much resources to give them, etc., by examining the problem, matching its characteristics to those of the Workers, and so on. Analogously to the Workers, the Tacticians also have a characterization of what they are good at, how costly they are, etc. And at the meta-meta-level there are reasoning modules called Strategists that decide which Tacticians to give control to, for how long, and when to interrupt them and switch to a different tactic (hand control to a different Tactician).

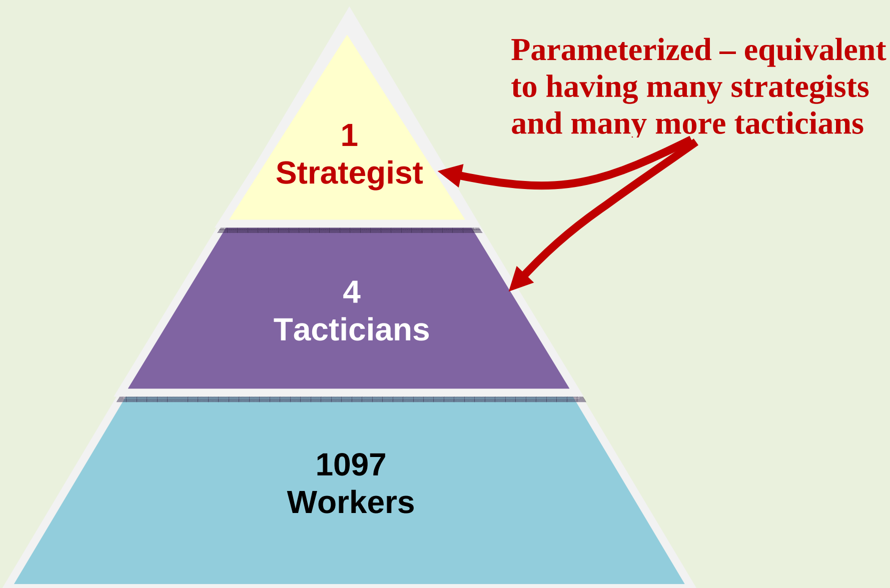


Figure 1: The architecture of the Cyc inference engine. This can also serve as a "harness" for other reasoning systems, which can be "published" as new Workers (proxies who just call the external reasoning system). These proxies can be characterized the same as the other 1097 Workers, so they can be called on when and as appropriate, to tackle sub-problems they would be good at, and can recursively re-broadcast sub-sub-problems back to the overall assemblage of modules – i.e., to the overall inference engine, hence they might end up being called on multiple times in the course of solving a single problem, with different sub*-problems.

2.1 Using Reinforcement Learning to Improve Tactical Decision-making

The first experiment was designed to determine the usefulness of harnessing certain kinds of meta-reasoners, namely those that use *reinforcement learning* techniques, to help a reasoner make tactical decisions.[1,2] The experiment analyzed data that had originally been collected for the DARPA Transfer Learning project[3]. As we will see below, this experiment showed that such meta-reasoners can be usefully employed for this meta-reasoning task, and indicated a methodology for employing them.

Cyc takes a hierarchical approach to inference: a ***strategist*** layer is responsible for selecting an appropriate overall approach to solving a given problem. As part of this strategy, any number of reasoning ***tacticians*** may be invoked, each of which uses a specific set of tactics for deciding what inference actions should be taken and when.

One can think of the basic reasoning engine as a set of cooperating agents, each of them capable of carrying out some specialized sort of reasoning (e.g., domain-independent transitive closure graph-walking; or domain-specific

"Derived from the psychological theory of the same name, in computer science, *Reinforcement Learning* is a sub-area of machine learning concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states. In economics and game theory, reinforcement learning is considered as a low-rationality interpretation of how equilibrium may arise.

The environment is typically formulated as a finite-state Markov decision process (MDP), and reinforcement learning algorithms for this context are highly related to dynamic programming techniques. State transition probabilities and reward probabilities in the MDP are typically stochastic but stationary over the course of the problem.

Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected." [2]

---

[1] A good grounding in Reinforcement Learning can be obtained from Leslie Kaebling's survey article available as http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html.

[2] This comes from a good, terse, recent introduction and overview of Reinforcement Learning, http://en.wikipedia.org/wiki/Reinforcement_learning . See also the references cited therein.

[3] Oblinger, D. 2005. Darpa Transfer Learning Program. This contains a good overview of transfer learning; available online as http://www.darpa.mil/ipto/solicitations/closed/05-29 PIP.htm .

balancing of aqueous chemical equations). The tacticians are a smaller set of agents which function as meta-reasoners: agents which guide the system as it decides which reasoners to call on and what resources to give them, when to interrupt them and switch directions, etc. And the strategists are an even smaller set of meta-meta-reasoners: agents which guide the system as it chooses which tacticians to listen to (e.g., when the tacticians disagree with each other).

Let's consider one of the most widely-used tactician modules in Cyc, the "balanced tactician". It balances the application of inference rules that simplify problems (which we call "***removal***" steps) with inference rules that make problems more complex (which we call "***<u>transformation</u>***" steps).

Why ever do "<u>transformation</u>" steps at all? I.e., why would one ever want to make a problem more complex? Consider a math problem like this one: If you multiply all the natural numbers from 3 to a trillion, and add one, is that number divisible by 15153241? The simplest way to solve this is to prove that more generally, if you multiple a set of numbers together and add 1, then that product has remainder 1 if you divide it by any of those numbers. Solving a harder, more general problem, is often a useful tactic. Of course *most* of the time one should be simplifying, doing "removal" steps.

In this experiment, a reinforcing learning (RL) tactician can be used to complement and inform the balanced tactician, identifying areas for potential improvement and suggesting strategies for achieving them.

As an example, the RL tactician task was set to improve *the time to first answer*. After training, the RL tactician was three times faster than the balanced tactician with respect to *median* time to first answer, although it was 36% slower with respect to *mean* time to first answer. Also the RL tactician suffered a 0.6% completeness loss compared to the balanced tactician. Out of 1648 queries run in this experiment, the balanced tactician answered 13 that the RL tactician failed to answer, and the RL tactician answered 9 that the balanced tactician failed to answer, for a net completeness loss of 4 queries. The <u>time to first answer</u> and the <u>number of queries that became answerable</u> were the two metrics used in the RL tactician's evaluation function; the <u>total time to find all possible answers</u> was not optimized for, in this experiment.

On a second set of simple test queries, the results were similar except the RL tactician was *faster* with respect to mean time to first answer. (It was still slower with respect to total time, but it was not aiming to optimize that metric.)

The results of this experiment are depicted in Figure 2.

In order to understand why one group of queries showed a substantial improvement, one first needs to understand a particular choice available to the Cyc inference Tactician when it attacks conjunctive queries. Consider, for example, a conjunction $A(x) \wedge B(x)$ that shares variables, x, across the two clauses. There are two ways the Tactician could attempt to solve this:

- JOIN: It could solve for all values satisfying A(*x*) and independently solve for all values satisfying B(*x*) and then simply intersect the two sets of results. This is called a "***join***".
- JOIN-ORDERED: Alternatively, it could solve only A(*x*) in its entirety and use the answers for *x* (call them $a_1, a_2, \ldots a_N$) to produce restricted forms of B(*x*) – i.e., $B(a_1)$, $B(a_2)$,… $B(a_N)$ – and then attempt to solve only that restricted set of (hopefully very small number of) of B(*x*) problems. This approach is called "***join ordered***" and makes sense when the estimated number of answers for B(*x*) is much larger than the estimated number of answers for A(*x*). When the two estimates are comparable, it makes more sense to solve both via a "join". (If there are far more A(x) answers than B(x) answers, the most appropriate tactic is to do a "join ordered" search in the opposite order.)

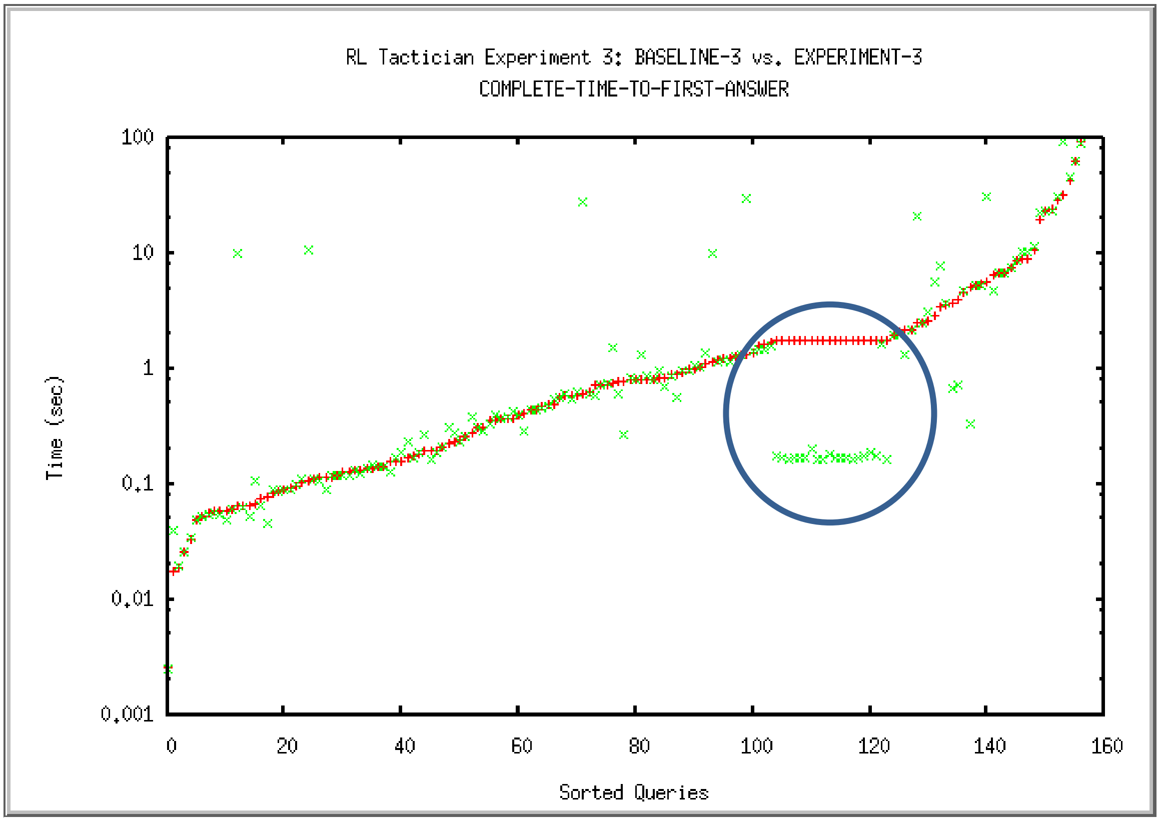


Figure 2: Time to first answer results for the RL Tactican (Green) and the Balanced Tactician (Red); the circle highlights a group of queries for which the RLT was 10x faster due to its use of join-order.

Practically the entire performance benefit seen in the above experiment was explained by one group of queries – the ones in the blue circle – with only one salient difference: the RL tactician chose a join-ordered search for their solution rather than a simple join. This proved effective because while ***joins*** are better for getting all answers effectively, ***join-ordered*** currently have better support for returning answers as they are found (i.e., "iteratively support") which reduces the time required to return the latency to first answer

(i.e., its "latency"). This, in turn, suggested an implementation enhancement for all tacticians, namely to make join tactics iterative.

While the particular inference improvement identified by this experiment was useful (i.e., the RL tactician automatically learned to sidestep the current lack of join tactic iteratively), the more significant result was demonstrating the use of an RL tactician as a means for identifying opportunities for modifying current inference tactics to improve specific performance objectives. I.e., modifying the inference engine so that joins are also iterative. It remains an open question as to whether there is a large number, or small number, of inference speed improvements that can be found by reinforcement-learning-trained inference engines. This question should be aggressively investigated in further work. The fact that the very first experiment we tried yielded such an improvement is very encouraging, and supports further investigation into this question.

2.2 Leveraging Prior Similar Queries: Cased-based Tactician

The second experiment was designed to evaluate another class of meta-reasoners, namely those that recognize when a query is similar to queries that have been asked in the past and then use the experience from answering those earlier queries to answer the current one more efficiently. This can be thought of as a form of analogical reasoning.

Inferences in Cyc and other theorem proving systems are rarely entirely novel; often queries are constructed by modification of an existing template, or an application that asks a number of similar queries or even the same query repeatedly. Given the relatively low cost of disk space, **storing** a large case library of past inferences is certainly viable. Given the relatively low cost of cpu "cycles", parallelizability, etc., **learning** from this stored corpus of past experience is also viable – i.e., getting the inference engine to learn how to better answer future queries, based on their similarity to past queries.

To draw a parallel to our own human educational experience, most of us studied calculus, and in particular the problem-solving method called integration by parts. After a while, we began to recognize which sorts of integration problems would and would not be attackable, solvable, that way. We characterized features of the problems that indicated the applicability of that tactic.

This approach led to the development of a **Case-Based Tactician** that, when given a new query, finds the most similar past queries and applies the approaches used to answer them to the new query.

There was already is a clear qualitative performance difference between Cyc's existing rule-only "proof-checker" mode (in which the set of rules to be used is already specified, allowing a very rapid search for a proof using those rules) and exploratory inference with all rules as candidates for inclusion. The Case-Based Tactician has managed to extend this sort of speed-up to novel problems, by remembering attributes of past cases of

successful proofs, and generalizing them (the tactics that did and didn't work well on those problems, historically) to brand new cases.

The case memory consists of proof recipes that encompass all Tactician and Worker decisions. This abstraction, or cached recipe, is called a "**proof spec"**. See Figure 3 for an example of proof spec generation, and see Figure 4 for an example of proof spec usage in action.

A proof spec is a problem-store-independent abstraction of the structure, modules and rules used to generate a proof. Proof specs can be generalized by replacing terms with generic patterns or by adding disjunctive options at any level of the proof spec. The case-based tactician based on this approach is referred to as a **Proof Spec Tactician**.

This approach is particular effective in situations where there the same query may be asked numerous times (such as "standing queries" in the intelligence domain) or where similar versions of a query may be asked often (such as those seeking the same information about many different people). For some applications, it should be possible to generate such queries upon application installation or startup, thereby greatly increasing the speed with which similar or analogous queries can be answered.

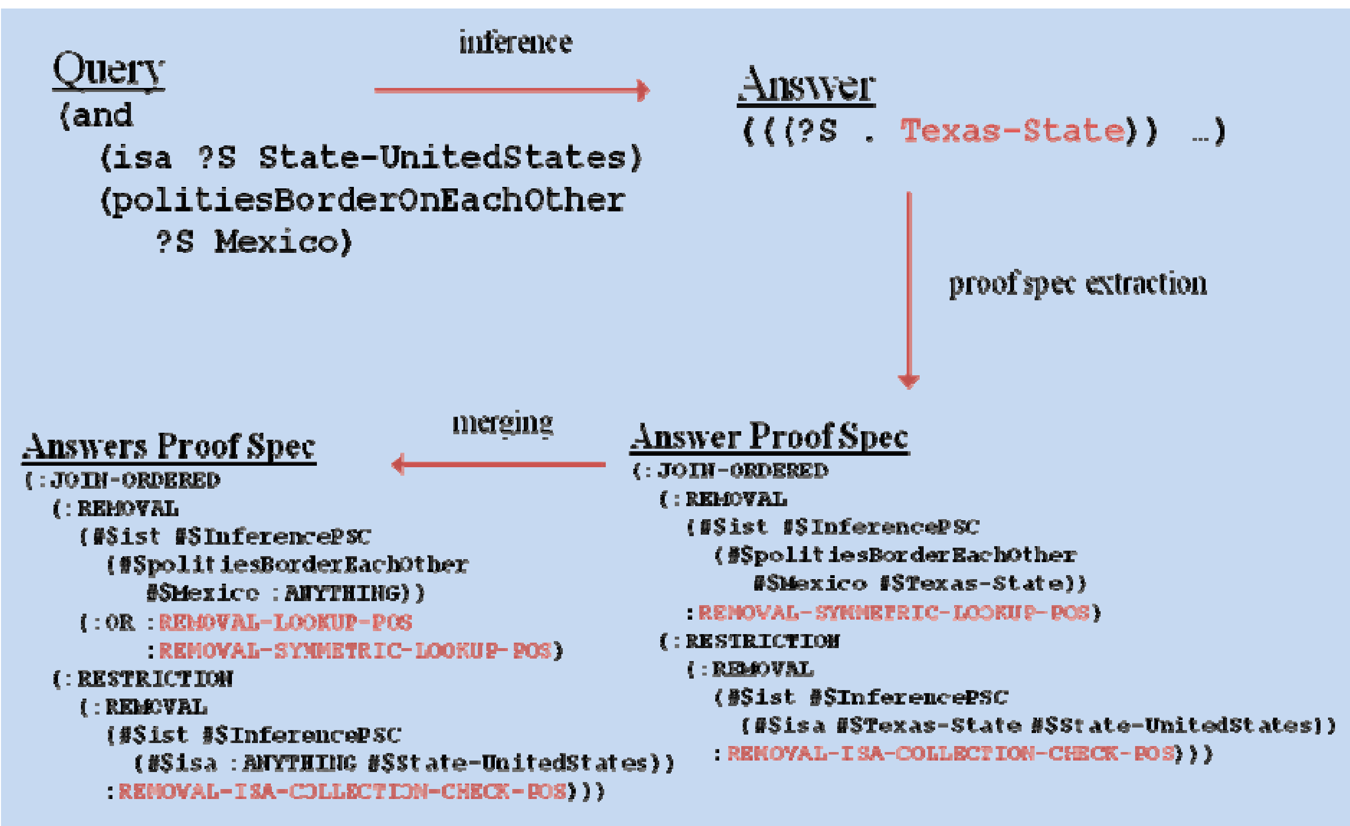


Figure 3: Generation of a Proof Spec from a set of answers to a prior query.

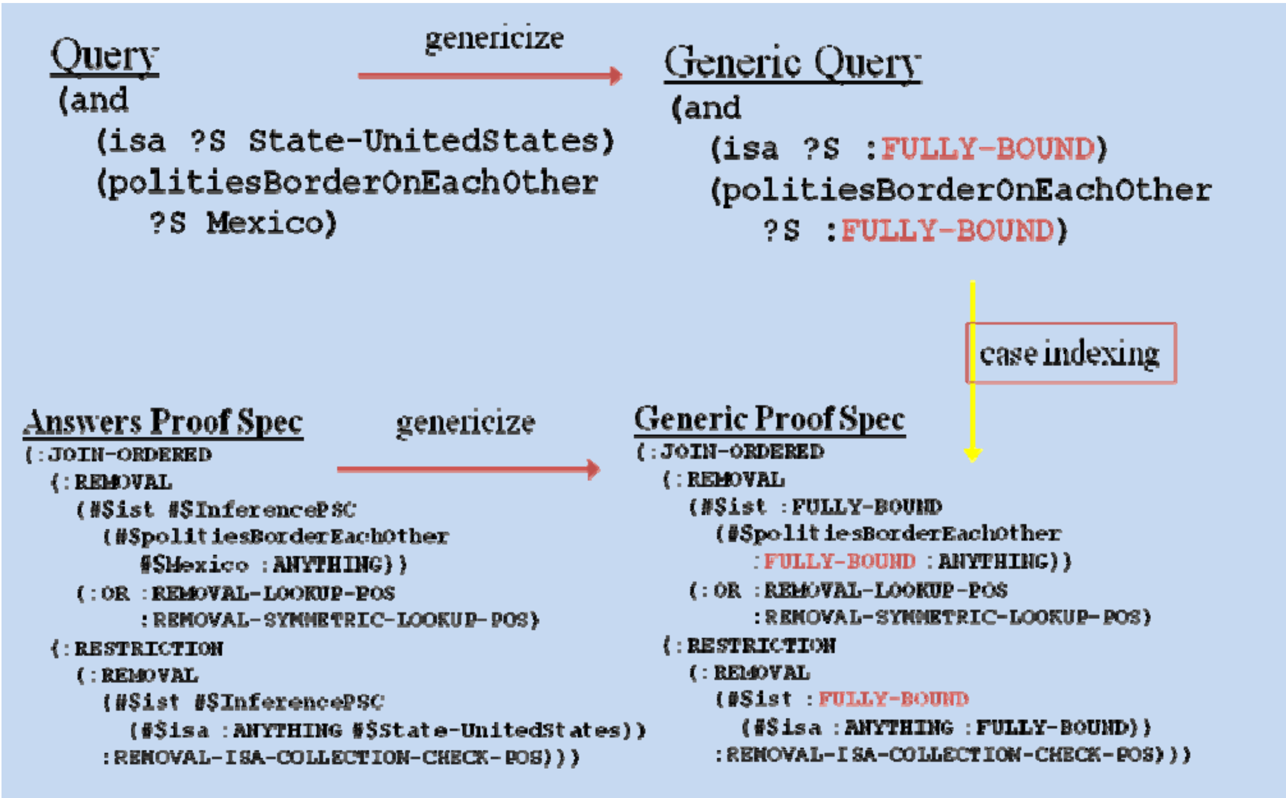


Figure 4: Use of a Proof Spec to increase efficiency in answering a new query. Once the Proof Spec from Fig. 3 was learned, queries such as "Which states in India border Pakistan?" could be answered efficiently.

To evaluate the potential impact of a Proof Spec Tactician, proof specs were collected for a large test corpus. Then, using a "straw-man" implementation of the proof spec (i.e., one that simply constrained the heuristic level (HL) modules (i.e., inference engines) and rules allowed to the tactician, but did not constrain it in any other way), these queries were re-run using these proof specs in order to assess their impact.

2.2.1 Answerability Analysis

The Proof Spec Tactician was permitted to say "Don't bother even *trying* a certain approach to solving this subproblem, I learned empirically from past experiences that it never works on this sort of subproblem." Because of this, the Proof Spec Tactician effectively constrains the reasoning that Cyc performs, and one concern was whether this approach would be less successful in answering queries. In particular, 838 queries were answerable in both the baseline and the experiment. 30 queries were answerable in the baseline but not in the experiment. 0 queries were answerable in the experiment but not the baseline. Thus, there was about a 3.5% reduction in the number of queries that were answered. However, the experiment resulted in about 10% more total answers than the

baseline did – presumably due to increased efficiency – so the net result was a 6.5% *increase* in answerability, not a net decrease.

We then analyzed the 30 queries that had become unanswerable. This experiment had been using a simplified, partial proof spec implementation intended to provide a lower-bound to performance improvement results. Looking at the 30 queries that ceased to be answerable with this partial proof spec implementation identified places where a full implementation of the Proof Spec Tactician would have succeeded, where this partial implementation failed. Thus, the efficiency analyses that follow included only the sub-corpus of questions for which the simpler proof spec implementation was representative.

2.2.2 Efficiency Analysis

A set of experiments were conducted running queries with and without the Proof Spec Tactician. A number of metrics were recorded covering two basic categories: completeness (as measured by the number of answers produced) and performance (measured in time until a first answers was returned and time until all answers were returned). For the performance metric, both mean times and medium times were calculated. Table 1 presents the results of the efficiency analysis using the mutually answerable subset of the corpus with and without the Proof Spec Tactician.

Table 1: Efficiency analysis of the mutually answerable queries showing the effects of including the Proof Spec Tactician

Analysis	Baseline	Experiment	Improvement Factor
SUM-ANSWER-COUNT	101620	110793	1.09
MEAN-COMPLETE-TIME-TO-FIRST-ANSWER	0.8002	0.4996	1.602
MEDIAN-COMPLETE-TIME-TO-FIRST-ANSWER	0.1367	0.06974	1.96
MEAN-COMPLETE-TOTAL-TIME	2.744	1.788	**1.535**
MEDIAN-COMPLETE-TOTAL-TIME	0.375	0.1641	2.285

These metrics show over a 50% speedup with respect to the mean and a greater than 200% speedup with respect to the median time to return all answers. It should be noted that these results are conservative in two ways: First, this speedup resulted from the straw-man version of the proof-spec tactician; an actual proof-spec tactician should do better still. Second, this speedup is for just for answerable queries; the proof-spec tactician can yield an almost infinite speedup on unanswerable queries by simply giving up immediately.

Figure 5 shows a graphical comparison of the baseline versus the Proof Spec Tactician results with respect to time to return all answers. In this case, the queries in each run were independently sorted by time so as to show an overall performance signature over the corpus.

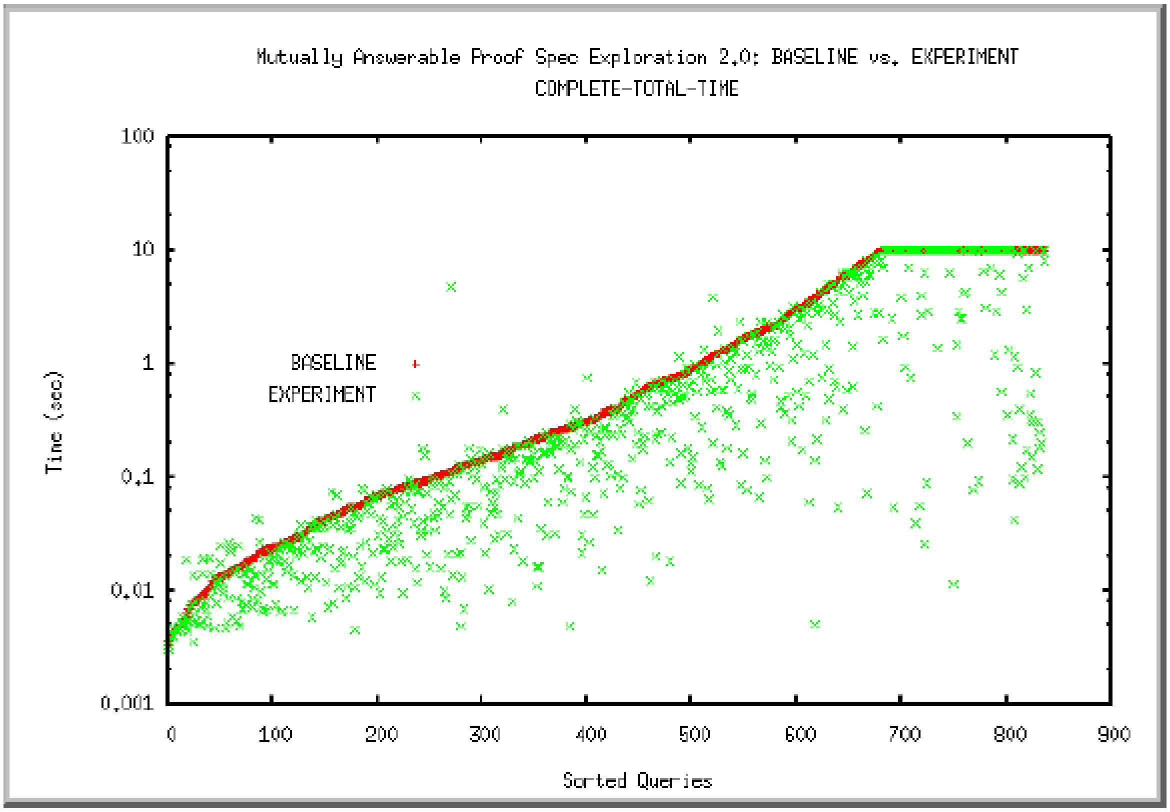


Figure 5. Comparison of time to first answer with and without the Proof Spec Tactician, with each set of results independently sorted by time.

Figure and 7 show the corresponding results for the time required to return the first answer, with the results independently sorted and then correlated, respectively.

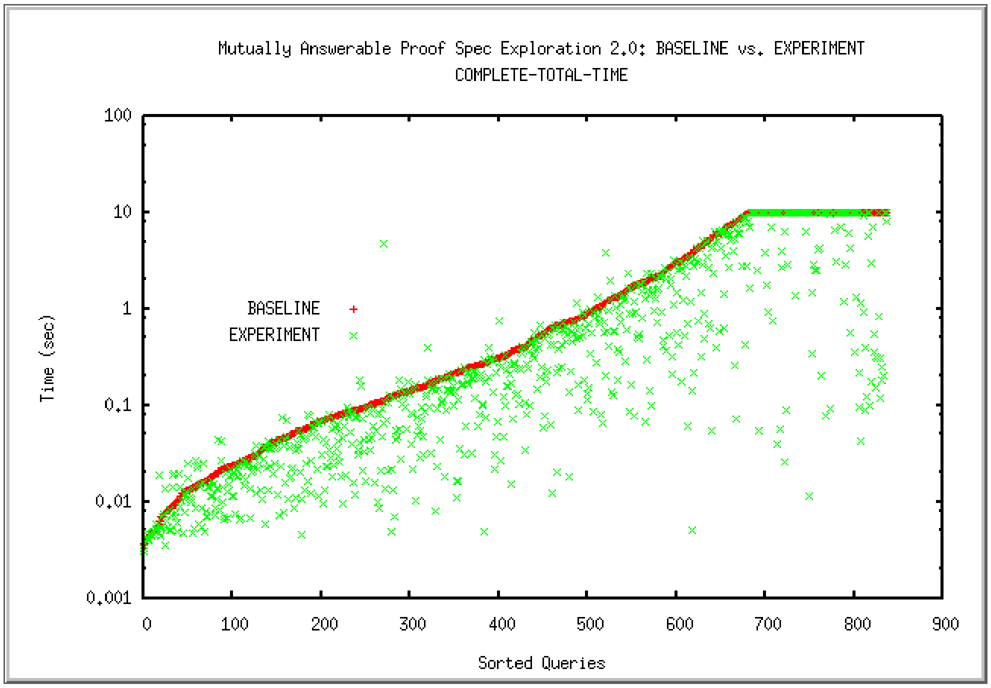


Figure 6: Comparison of <u>time to first answer</u> with and without the Proof Spec Tactician, with each set of results independently sorted by time.

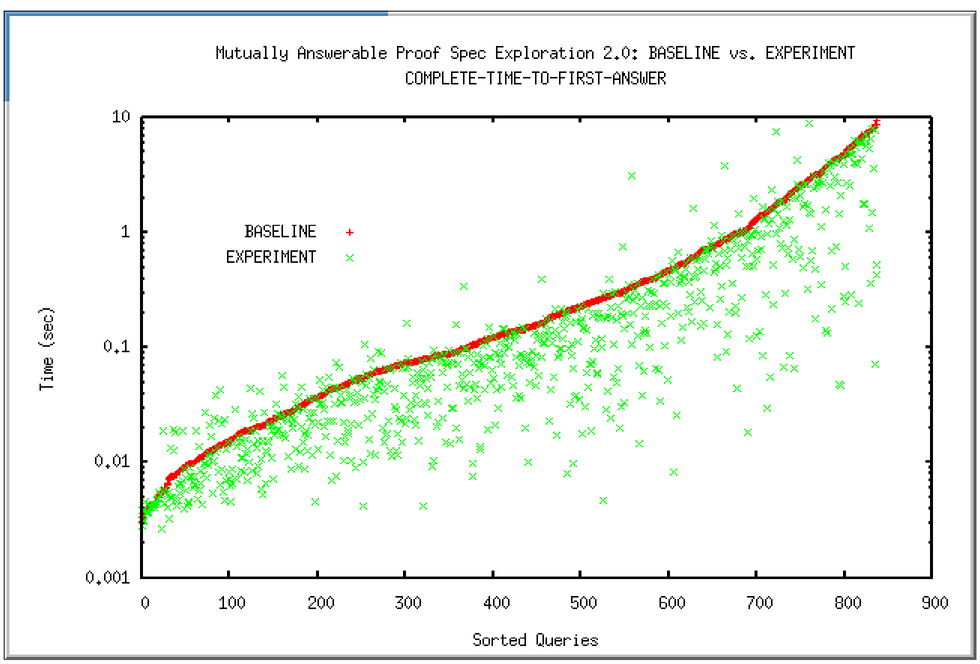


Figure 7: Comparison of <u>time to first answer</u> with and without the Proof Spec Tactician, showing the correlated results for each query

2.2.3 Final Proof-Spec Results

Based on the promising results from the straw man (partial) implementation of the Proof Spec Tactician, we implemented a real, full Proof Spec Tactician that takes a proof spec provided as a query parameter and ensures that only tactics that rigorously match the proof spec are allowed to be executed. Since the Tactician already maintains bookkeeping information to keep track of which problems in a problem store are relevant to a particular inference, it was natural to extend the notion of relevance to include explicit recording of the relevant proof specs for a problem. Relevant proof specs propagate down problem links based on the types of the links in a way exactly analogous to relevance propagation.

We evaluated the query-genericizing algorithm performance by investigating an automatically generated corpus of a large number of analogous queries from a representative setting, namely all the backward inferences that are performed by the forward inference algorithm when a new assertion triggers forward inference. Forward inference makes extremely heavy use and is dominated by backward inference. Furthermore, since the same set of forward rules are being repeatedly triggered by KB modifications, the queries that result are likely to be almost totally unique (due to the new content triggering the inference) but almost totally analogous to other recent queries (due to the same rule being triggered).

We obtained 668 queries by running through a randomly chosen transcript of queries and operations actually being asked of Cyc (namely, the initial portion of the KB 941 modification transcript). Only about 30% (194) of these queries had unique genericized query forms and 71% (474) had the same generic form as some other query.

We evaluated the performance on two corpora, selected from previous Intelligence Community applications, the Arete TKB (terrorism knowledge base) corpus and the Leviathan corpus.

2.2.3.1. Arete Corpus. The Arete TKB corpus contains queries that typically involved conjunctions requiring relatively shallow inference. This corpus was designed to mirror the types of queries intelligence analysts perform when performing investigative analysis on suspected or known terrorists, and organizations of interest. For each query in the baseline run, the proof spec for its result was generated and passed to the proof spec tactician in the experiment run.

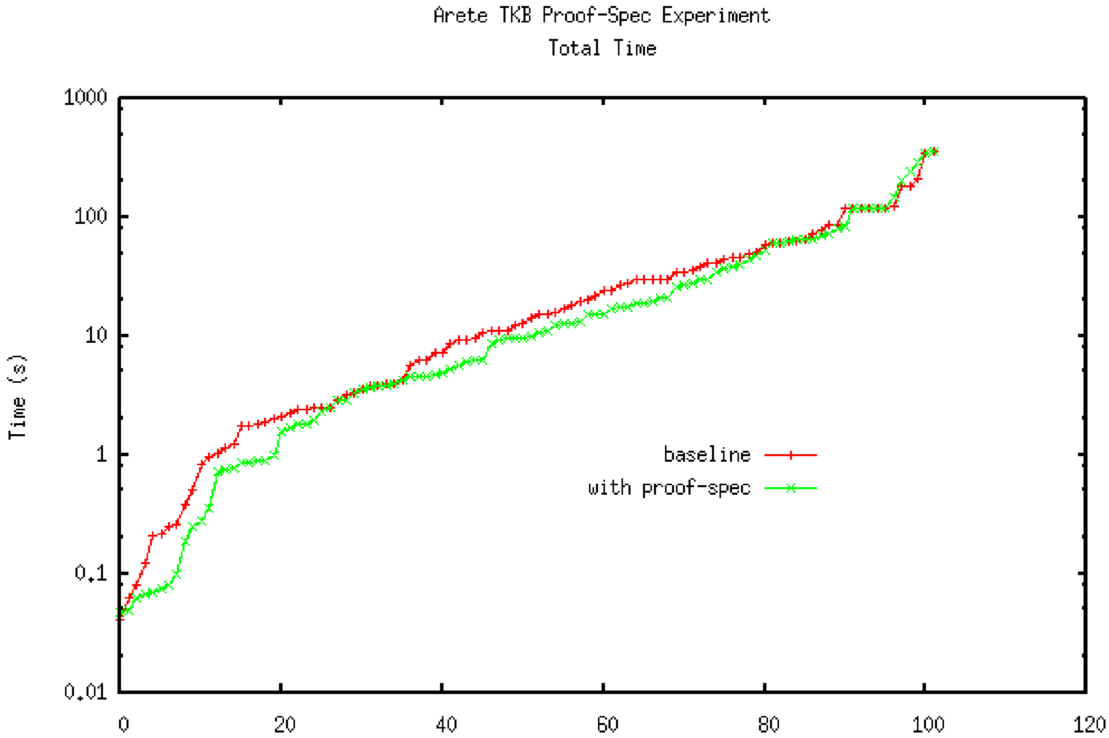


Figure 8. Comparison of <u>total time</u> with and without the Proof Spec Tactician, with each set of results independently sorted by time.

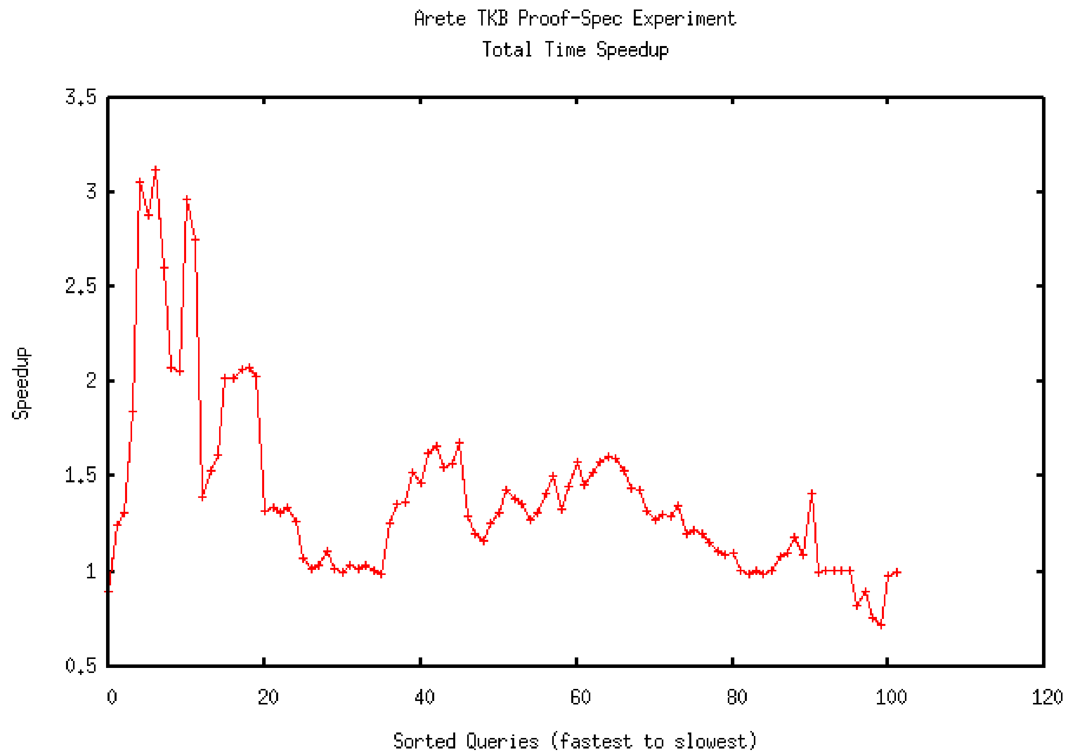


Figure 9. Speedup signature of the Proof Spec Tactician with results independently sorted by time.

For this corpus, the speedup appears primarily at the faster (leftmost) end of the spectrum, with only a moderate speedup at the middle and slower ends of the spectrum.

2.2.3.2. Leviathan corpus. The second corpus we evaluated was the Leviathan corpus, which contains deeper queries that require chaining two or more rules together. This corpus is designed to mirror the types of queries that appear in the more analytical investigations done by analysts when pondering scenarios and hypothesizing possible links between agents and organizations. These queries tend to be longer and generally suffer more from the algorithmic effects of exponential fanout of the search space when chaining rules together.

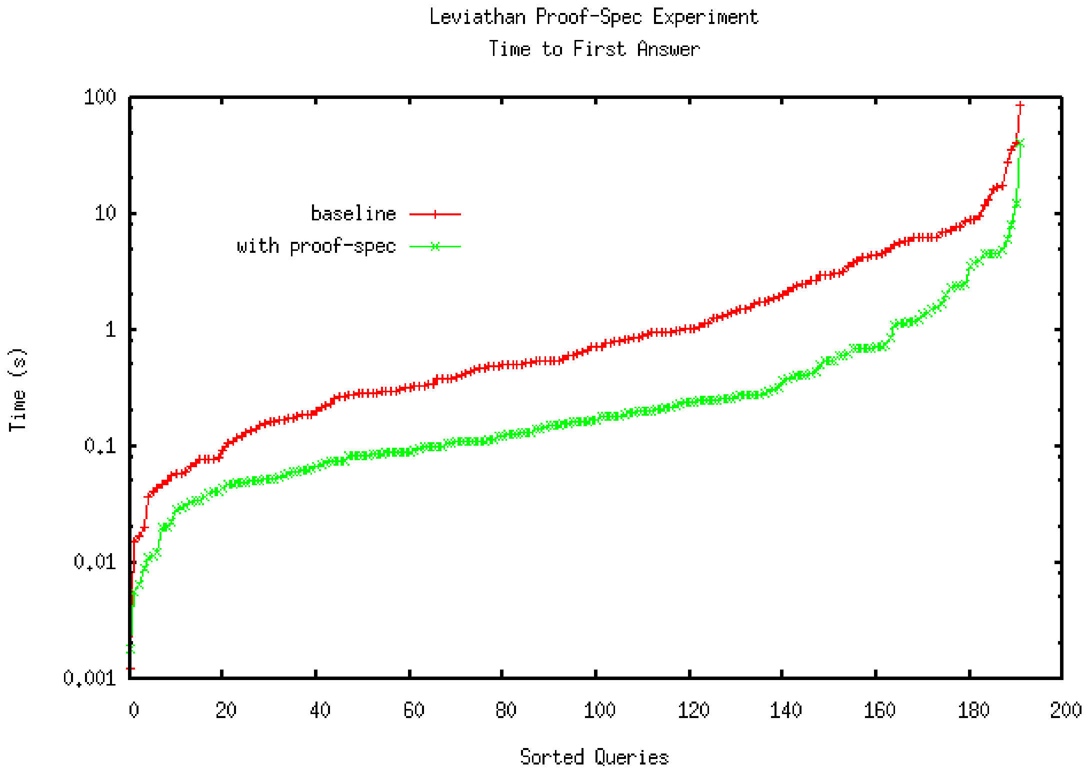


Figure 10. Comparison of <u>time to first answer</u> with and without the Proof Spec Tactician, with each set of results independently sorted by time.

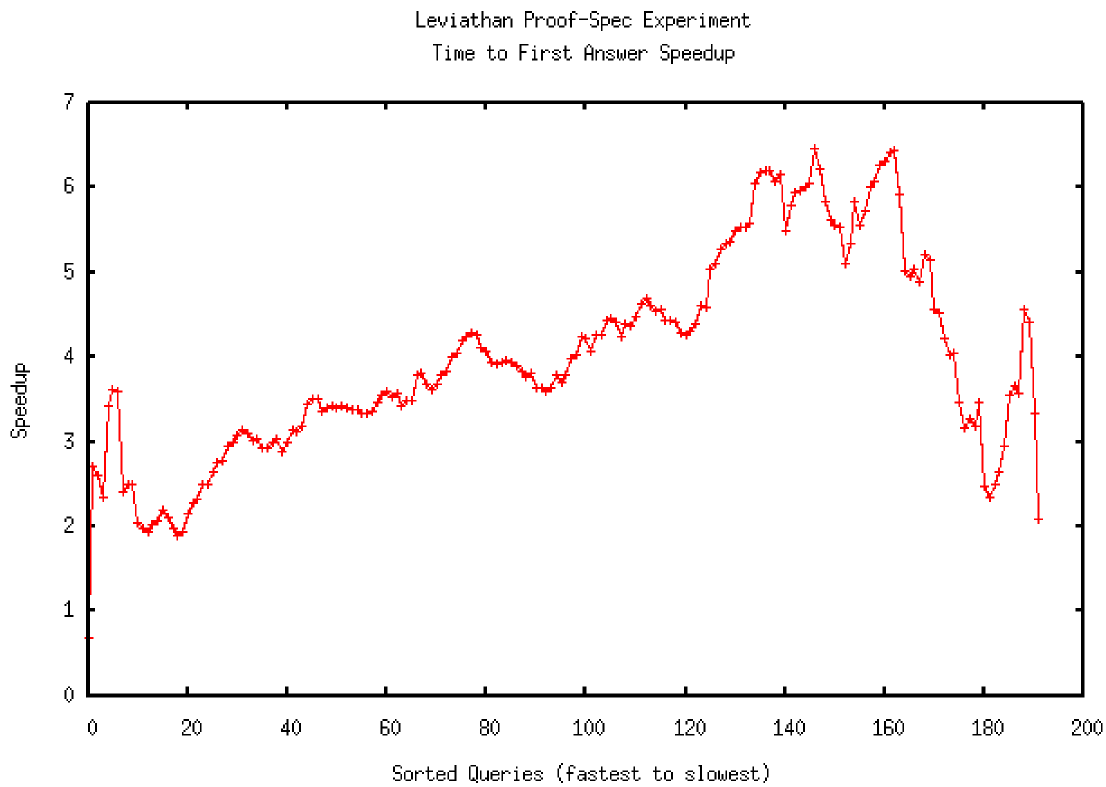


Figure 11. Speedup signature of the Proof Spec Tactician with results independently sorted by time.

For this corpus, the speedup is significant and across-the-board, with even greater benefit for the harder queries at the slower (rightmost) end of the spectrum.

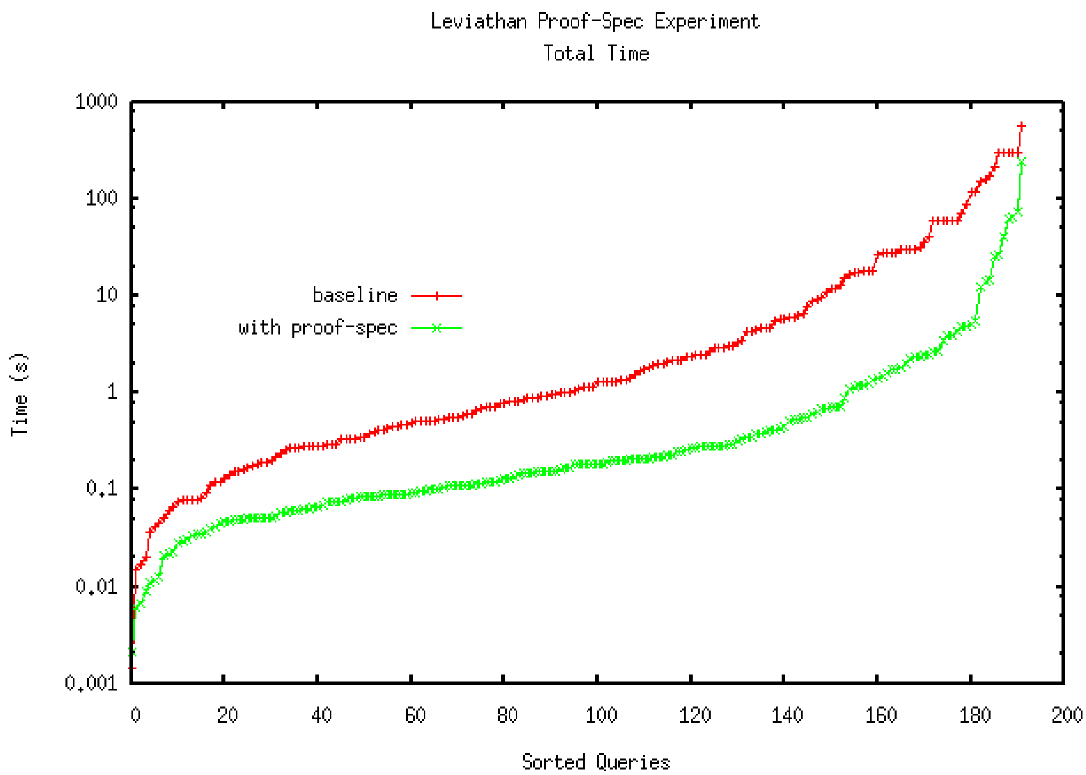


Figure 12. Comparison of <u>total time</u> with and without the Proof Spec Tactician, with each set of results independently sorted by time.

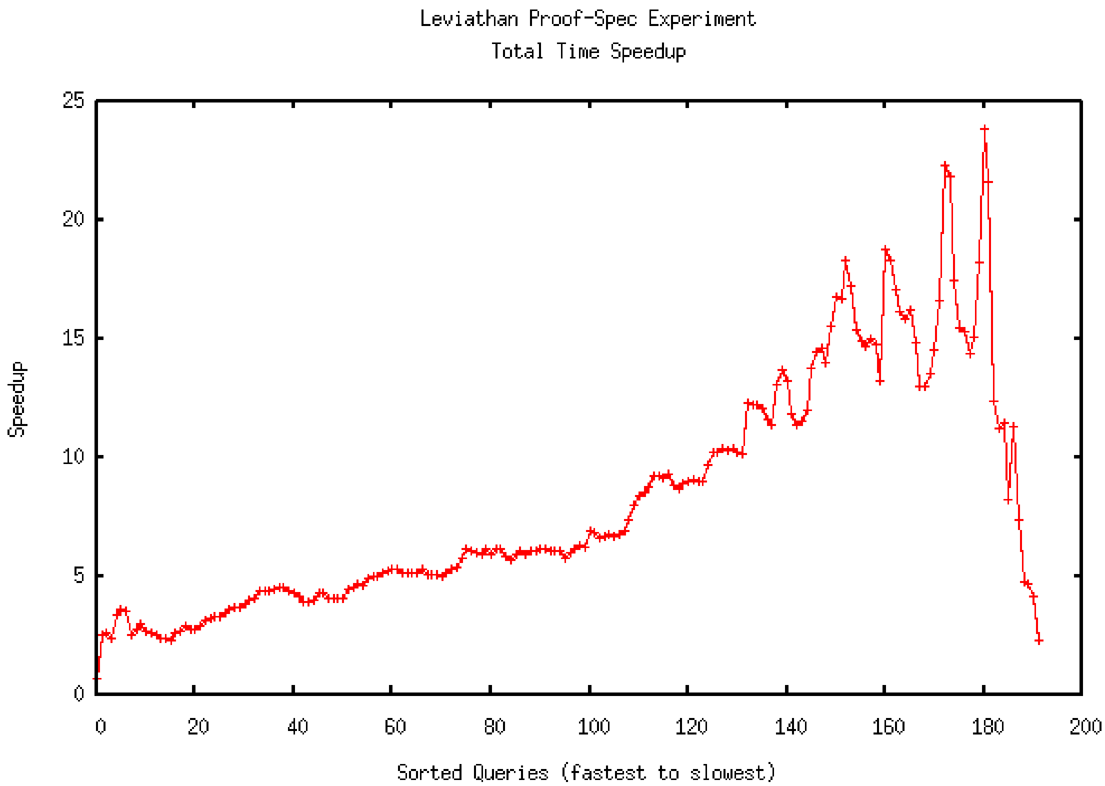


Figure 13. Speedup signature of the Proof Spec Tactician with results independently sorted by time.

For total time in the Leviathan corpus, the Proof Spec Tactician results in significant speedups across the entire spectrum, with large speedups at the slower end of the spectrum.

The speedup of the very fastest and very slowest queries with only moderate speedups on the middle queries was unexpected. We surmise that for the fastest queries the benefit comes from avoiding costly but less fruitful reasoning modules and that the benefit for the slowest queries comes from algorithmic simplification through pruning the fanout in the search space. Note also that the benefit for the middle queries displayed by the straw man Proof Spec Tactician implementation appears to be missing in the actual Proof Spec Tactician. We believe this is due to inefficiency in the current proof spec propagation code, and that an optimization pass would recoup most of this loss. If not, then a clear hybridization opportunity exists by using the straw man approach for the faster queries and the complete proof spec approach for the longer queries.

2.3 Knowledge Clustering

The third experiment explored ways of clustering knowledge so as to much more effectively predict which parts of the KB will be relevant to a given query, thus reducing the size of the search space for answering the query as well as reducing the amount of knowledge sent to an external reasoner to help with the query. The latter could become a dominant factor in harnessing E, SPASS, Vampire, and most other theorem provers.

For this experiment, a test suite of queries whose proofs included two or more rules, called the Leviathan corpus, was selected as the baseline. From a baseline run of this corpus, all pairs of rules that ever actually successfully co-occurred in any inference proof together were identified. This data was used to construct (and dynamically maintain) an undirected graph, called the "Rule Connectivity Graph", comprising all such pairs of co-occurring rules.

Next, using this graph, a new inference transformation heuristic was added that favored proof paths that involved sets of rules that had the highest fraction of pairwise graph connectivity. (See 14 for an example.) The hypothesis was that this heuristic would greatly improve the *time to first answer* for the Leviathan corpus (over a blind subset of this corpus) by heuristically delaying historically unpromising rule combinations and effectively reducing the branching factor among the heuristically promising portions of the search space.

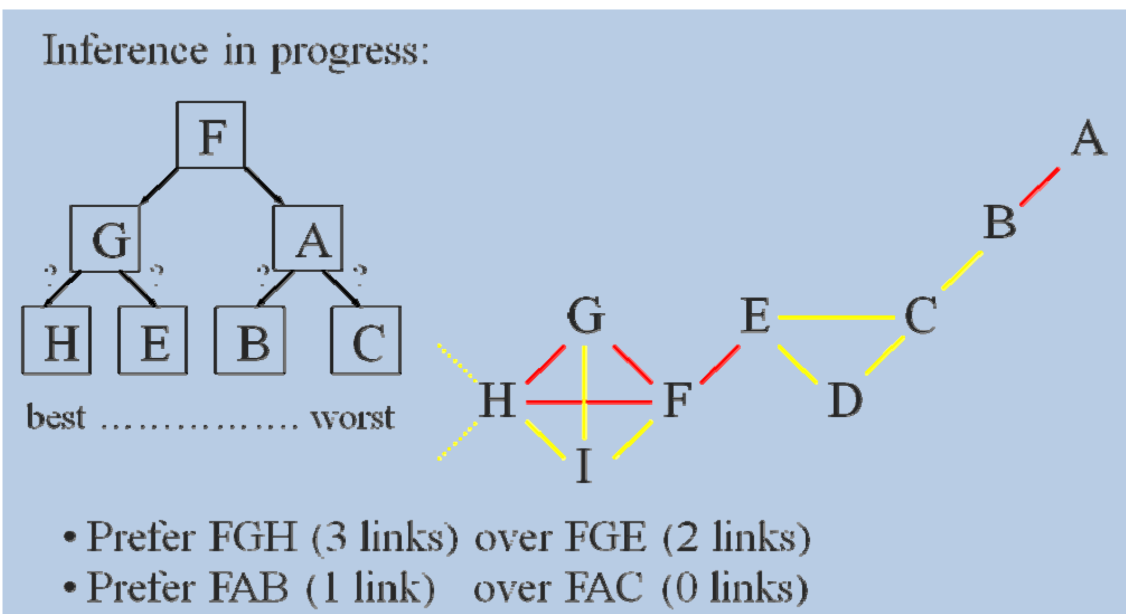


Figure 14. A rule co-occurrence heuristic favors solution paths that include pairs of rules that have previously co-occured on another proof (e.g., F&G, F&H, G&H, or A&B).

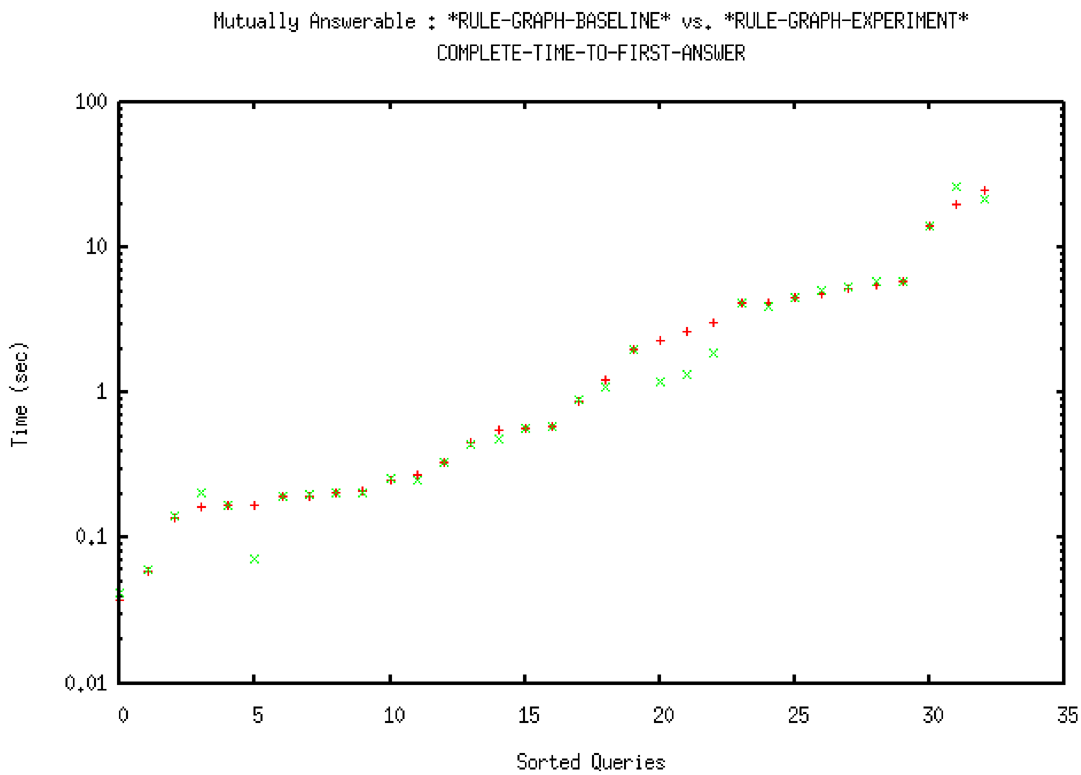


Figure 15. Comparison of <u>time to first answer</u> with (green) and without (red) the rule connectivity heuristic, showing the correlated results for each query

The empirical results of this experiment were surprising and frankly a bit disappointing. In fact, without the caption (of Figure 15) explaining which was the red curve and which was the green curve, it would be difficult to tell which was the baseline and which was the result of using the rule connectivity heuristic. The heuristic appears to have very little additional benefit beyond what's already available in the baseline. Only a small group of queries in the middle of the signature improved.

We believe it is worth investigating why it was not more beneficial We surmise that the faster queries may naturally have fewer ways of combining the rules and therefore don't need the heuristic suggestion of candidate combinations. We surmise that the slower queries are being dominated by some other algorithmic issue that warrants further investigation. Another possible explanation is that the weight of the heuristic was not high enough to have a sufficiently discriminating impact on search ordering.

Still, the potential promise from algorithmic reduction of the search space using this approach suggests that further exploration into meta-reasoning about other promising

proof skeletons is warranted and that different type of external reasoners suited to this purpose may be appropriate.

Despite this one disappointing result, the results from the three families of experiments (Sections 2.1 – 2.3) were overall quite dramatic, and together resulted in more than a factor of 10 speedup of the system for a significant reasons (i.e., ones which will be generalizable to other domains and to other theorem proving systems.)

That brings up the second portion of the project, this past year: engaging the broader international theorem proving community, so that in the future they will continue to work on this characterizing and harnessing notion themselves.

3. Challenging the Theorem Proving Community

One of the major goals of this project was to show how multiple theorem provers could be combined to achieve a theorem-proving capability that is superior to that of any individual theorem prover (including Cyc itself). The most naïve approach would be to examine a particular problem to be solved (or query to be answered) and then select which theorem prover is best suited to address it. Previous work showed us that this was sub-optimal: various provers do well on various subproblems and poorly on other subproblems. [This particular solution would have been impractical for an even simpler reason: none of the other theorem provers we experimented with were able to load the entire Cyc KB due to its size!] .

To address this need, two problem suites were produced: A *scaling suite* and an *elaboration suite*. The scaling suite contains knowledge bases of varying sizes to enable the theorem proving community to address at least scaled-down versions of Cyc-like problems, even if they are unable to load the full KB. The elaboration suite focuses on problems that, in going from one problem to the next, add or remove things from the theory, encouraging the theorem problem community to address such deltas as deltas, rather than resorting to reloading the entire theory from scratch for each problem.

Below we report on the techniques we used to generate these problem suites and how they are already stimulating the research community to try to overcome the problems the currently have addressing them. Next, we report on the integration feasibility experiments we performed.

3.1 New Classes of TPTP Problems

The 21st Conference on Automated Deduction (**CADE-21**) was held in Bremen, Germany from July 15 - 20, 2007. CADE is the major forum for the presentation of research on all aspects of automated deduction. We have been using this project as a springboard for increasing the visibility of large scale, IC-community-relevant inferential problem solving in the theorem-proving community worldwide, and in the TPTP community in particular. As a result, Cycorp was invited to speak at the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories (**ESARLT**) which was held on July 15th, and we took them up on this offer (no foreign travel was charged against the budget of this project). The workshop proved to be an excellent forum to expose the wider automated deduction community to the inference design challenges posed by Cyc and large scale inference.

Keith Goolsbey presented an invited talk there, entitled ***Cyc Design Challenges and Solutions*** which had the following abstract:

> Cyc comprises a large, contextualized, common sense knowledge base (KB) which is encoded in an expressive representation language (essentially FOL with a few key extensions) and paired with an inference engine optimized for the classes of queries we most frequently encounter. These queries tend to mix relatively shallow reasoning within one of a large number of idiosyncratic subtheories with relatively deep reasoning within one of a very small number of stylized subtheories. The constraints of these queries in a large and expressive KB combined with the need to efficiently react to KB elaboration together provide a unique set of design challenges that are extremely stressful for the solutions provided by the current state of the art FOL theorem provers. The solutions to these challenges currently adopted by the Cyc inference engine will be presented within the context of a new suite of TPTP problems that are derived from Cyc's KB and typical queries and are intended to demonstrate Cyc's design challenges for investigation by the wider community.

To recap, the TPTP Problem Library is a collection of test problems for automated theorem proving (ATP) systems. This is the main regression and benchmarking suite used by the ATP community worldwide (primarily comprised of researchers in Europe, the U.S., Australia, India, China, and Japan) and as such it acts as the primary use-cases on which this community focuses its efforts. Everyone acknowledges that many of the problem sets in this library reflect the mathematical theorem proving historical background of early ATP efforts, namely relatively deep general proofs within extremely small but powerful theories. This also reflects what could be done "by hand" early in the 20^{th} century, and what could be done by early computers in the 1950's-1980's.

The Cyc use-cases – and the general IC pathfinding problem – are very different from those narrow reasoning problems in which the entire small KB changes every time. Namely, our use-cases involve an extremely large KB that changes "a little bit" between queries, or possibly not at all, and which is targeted by many queries each involving relatively shallow proofs. Problems with that characterization were completely unrepresented in the TPTP library. Not surprisingly, without focus on Cyc-like theories this community is not directly confronting Cyc-like design problems or producing novel, innovative solutions for these problems. To eliminate this blind spot, we developed a new TPTP problem set derived from OpenCyc content and Cyc-representative queries.

The new TPTP problem set also addresses the problem of other theorem provers not being able to load a KB the size of Cyc's. Rather than each problem requiring all of Cyc's knowledge, we constructed six different 'levels' of problem, each containing progressively larger theories:

- **Level 1: Proof-only -** In the Level 1 or "proof-only" problems, we ran a query against Cyc's theorem prover and extracted all and only the facts necessary to answer the problem. Just those facts were included in the theory provided for this problem. This is close to what the existing TPTP test problems are like, and was meant to ease the ATP research community into working on our task, enabling them to succeed on some of our problems rather than just starting with the hard ones and giving up.

- **Level 2: All Known useful facts** - In Level 2, the background theory contained all of the facts used in the proofs of any of the fifty problems generated. So for any problem (in this set of 50), all facts in the theory were known to be useful somewhere, but not necessarily in this particular problem. This eased the ATP research community into the mindset of having a large set of queries going against a single theory, a single axiom set, without having too astronomical a number of irrelevant axioms; i.e., in this case, about 10% of the axioms would turn out to be used in proving each problem (some axioms were used in proving more than one problem).

- **Level 3: Addition of some Potentially Relevant Facts** - In Level 3, the background theory includes all information from Level 2, plus some facts not used in any proofs, but where all the predicates (relations, functions, properties, attributes, slots) *were* used in *some* proofs. So there weren't any predicates that were just red herrings here, and there weren't too many irrelevant axioms even involving those used-somewhere predicates.

- **Level 4: Addition of all Potentially Relevant Facts** - In Level 4, the background theory includes all information from Level 3, plus all facts in the current Cyc KB using predicates that are in any of the proofs. So as with Level 3, there were no red herring predicates, but the number of axioms involving some of the predicates present could be quite high – up into the tens of thousands of axioms in many cases.

- **Level 5: Addition of some Irrelevant Facts** - In Level 5, the background theory includes all information from Level 4, plus a random set of additional assertions that are completely unrelated to any of the problem queries. This is much more representative of the IC pathfinding problem: separating the wheat from the chaff.

- **Level 6: Entire KB** - In Level 6, the entire OpenCyc KB is included as the background theory, adding literally a million axioms which were (in hindsight of course) irrelevant to proving any of the test problems.

3.1.1 TPTP Problem Generator

There has been a lot of impressive work in the First-Order Logic (FOL) theorem proving community in recent years. Much of this advance can be credited to the community adopting a standard test representation language and accumulating a large repository of tests in this representation (TPTP) via yearly, objective competitions (CASC). Given the similarity of expressiveness of FOL and CycL, it appears at first blush that the champions of these competitions (VAMPIRE, E, and SPASS) might be ideally suited to providing commonsense inference capabilities over the Cyc knowledge base. However, analysis of the classes of problems in the competitions reveals an apparent impedance mismatch between the theory expectations of the current state of the art theorem provers and the theory expressed in the Cyc KB.

FOL problems have historically been driven by mathematical use-cases in which a single relatively deep proof within a single, small, static theory is obtained. On the other hand, commonsense queries in Cyc tend to be a very large number of queries instantiated from a relatively large set of query classes targeting a KB with a large number of different (micro-) theories, which can dynamically change as the KB is updated. Furthermore, these queries usually expect multiple answers to be derived and justifications for these different answers to be maintained. Finally, the commonsense proofs tend to be either relatively shallow or deep but only within a very stylized proof space.

Because of these potential differences in the expectations of the theorem provers, it was not initially clear whether current theorem provers, developed largely to address problems in mathematical logic, would perform well on Cyc commonsense inferences. Our preliminary experiments (with Cyc itself – see Section 2 above – and with the other theorem provers -- see e.g. Section 3.2.1 below) show that current theorem provers have substantial difficulty with the Cyc problem set. Below we discuss the methodology we employed when devising the test set, which we used for our preliminary experiments and also presented at the ESARLT workshop discussed above. It is a goal of this work on TPTP, and indeed of the Efficient Pathfinding project in its initial conception, to inspire the theorem-proving research community to address the classes of queries and theories over everyday concepts that have been the focus of work on Cyc. By engaging the TP-community it will be possible to perform 'apples to apples' comparisons with respect to performance, and to properly leverage the design and implementation benefits that this community can provide in applying inference to problems of interest to the IC.

3.1.2 Requirements

The design constraints that Cyc has been developed under have been driven primarily by the intelligence community in the past decade; they are substantially different from those that guided the design of most theorem provers.

The six constraints that we paid specific attention to when designing the TPTP problem set include the following:

1. **Expressivity**: Cyc includes a number of extensions to first order logic which can be converted into a first-order logical representation. However, when these extensions are expressed in first order logic, care must be taken or they can overwhelm other theorem provers. One substantial problem posed by Cyc's expressivity is the need to canonicalize queries and facts as they are presented, rather than assuming everything is already in Conjunctive Normal Form. By presenting all of our TPTP problems in FOL, all systems are required to factor in canonicalization time. Various other aspects of Cyc's language (e.g. micro-theory

inheritance, functions, quantification into predicates) require special handling to do efficiently.[4]

2. **Large Background Theory**: Because most theories used by theorem provers are fairly small, load time can be effectively ignored. However, with large background theories, the effect of load time cannot be ignored. Substantial amounts of time may be spent pre-seeding the theorem-prover with useful information from the theory if there is a large amount of background information. Theorem-provers that go to considerable effort at load time, for example in indexing theorems, may be shifting work to load time that would need to be done at run time if the theories were larger.

3. **Multiple Varied Queries**: The fact that theorem provers may be queried multiple times is reflected in the fact that our TPTP problems include multiple queries to be posed against the same theory. A theorem prover with persistent state, for instance, they might be able to maintain this background theory and reuse it against the working clauses. This type of behavior should be encouraged. Systems that rely on modifying their knowledge bases as part of answering a question will be confronted with the challenge of figuring out how to ask multiple queries during a single run.

4. **Microtheories**: Most reasoning systems do not allow for knowledge inheritance, so if multiple reasoning contexts are needed, each context must effectively contain all of the knowledge that logical belongs in it. This is potentially problematic if hundreds or thousands of reasoning contexts are required. Cyc contains a context (or *microtheory*) mechanism in which microtheories are first order objects and may subsume other microtheories. All assertions in Cyc are stated within a microtheory. To capture this within a first-order representation, we transformed statements of the form *(ist M_1 R(X))* – meaning that the statement R(x) is true in the context M_1 – into the statement *(mtVisible M_1)* $\rightarrow$ *R(x)* – meaning that if context M_1 is "visible", then statement R(x) is true – and a rule stating that if M_x is visible, the parent context of M_x is also visible. Assertions that particular contexts are visible – *(mtVisible M_1)* – are then included as part of the conjecture.

   When boiled down to a first-order representation, knowledge inheritance (in the form of mtVisible assertions) effectively prohibits the use of unlimited forward propagation because of extremely large number of mtVisible assertions that are necessary. Systems that rely on forward propagating all facts run the risk of

[4] For more details on these higher-order features, see First-Orderized ResearchCyc: Expressivity and Efficiency in a Common-Sense Ontology by Deepak Ramachandran, P. Reagan, and K. Goolsbey in *Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications.* Pittsburgh, Pennsylvania, July 2005.

spending so much time forward propagating the mtVisible facts that they never get around to dealing with the actual problem.

5. **Elaboration Tolerance**: The requirement that the reasoning system be mutable at run-time presents several challenges to theorem provers. In particular, it means that the system cannot safely ignore load time, the language cannot remain fixed for the entire run-time of the system, the addition of new assertions means that the system cannot simply be restarted for updates (because assertions made before the update would be lost), and the fact that some assertions and concepts will be removed means that the systems need to have facilities maintaining system integrity with respect to the consequences of removal (known as “truth-maintenance”).

6. **Queries Interspersed with Elaboration**: A system that is required to perform interspersed theory extension and queries must be able to do both of those within a reasonable time-frame. Performing KB compilation after elaboration may cause significant delays between when the KB gets updated and when the system is able to answer queries against the KB, especially if there are many queries interspersed in the new assertions, and if the KB compilation is time-consuming.

3.1.3 TPTP Problem Creation

The TPTP elaboration suite was designed to simulate a set of KB operations:

- Creates: the addition of new terms;
- Asserts: the addition of new statements about existing terms;
- Queries: accessing information already in the KB;
- Unasserts: removing existing assertions from the KB; *and*
- Kills: removing existing terms from the KB.

To do this, we took advantage of the "include" feature of TPTP, and also add additional "pragmas" (pragmatics) in the form of specially formatted TPTP comments. The TPTP format already provided for structured comments that act as meta-statements; we negotiated with the TPTP community to gain acceptance to use these for purposes such as representing which parts of the KB are static and which dynamic, and for identifying a sequence of problems and ordering the problems with that sequence (as opposed to just having a set of individual problems).

Using these meta-statements, the OpenCyc KB was split up into static and dynamic sections; the dynamic section was further split into those changes that had no impact on query answerability (so called “dynamic chaff”) and those that changed whether a query was answerable (so called “dynamic linchpin.

The static KB forms one massive TPTP include file. This include directive is preceded by a pragma indicating that this is the static KB, and the theorem prover can assume that it will be present throughout all the Cyc TPTP problems. The dynamic portions of the KB are separated out into many different include files via a method described below. Each TPTP problem in the problem set consists of:

- a header
- the static KB ***include*** directive
- several additional ***include*** directives (to pick out a subset of the dynamic KB)
- the conjecture

An ***include file*** that is present in one query but absent in a following query simulates a set of unasserts (i.e., removing statements about terms) and kills (i.e., removing the terms themselves). An ***include file*** that is absent in one query but present in a following query simulates a set of asserts and creates.

In order to create the problems, we generated synthetic KB content. This was necessary for elaboration and also because we wanted these problem-sets to be runnable with OpenCyc, which has a relatively small number of implication rules and a relatively shallow set of predicates. The process of choosing the dynamic KB content and the conjecture for each problem is described below.

To start out, we generated many TPTP problems that test aspects of inference, but do not test elaboration, called *non-elaboration queries*. To generate these, we called the *query generation modules* directly (described below). We generated equal proportions of answerable and unanswerable queries, and for those queries where it made sense, we generated equal proportions of open queries (where the system expects to get multiple bindings) and closed (Boolean) queries. We analyzed and indexed the justifications of the answerable queries, then set them aside for a while.

Next we identified a set of *linchpin* CycL terms. This was done by calling a set of *linchpin modules*. Each linchpin module describes a particular kind of inference and a particular kind of elaboration that we want to test. A linchpin module calls its associated query generation module (described below) to generate an answerable TPTP problem. It chooses its linchpin from the full justification/proof of that answerable problem, making sure that it's not used in the justification of any other problem. If linchpins are found, it throws the problem away and tries again until it succeeds. These are called "linchpins" because each one is a critical part of the justification of exactly one answerable TPTP problem. Each linchpin module generates at least 3 linchpins.

For each linchpin module, the system then partitions its linchpins into three *elaboration categories*: static, to-be-added, and to-be-removed. Static means that the linchpin will be present throughout the simulation. To-be-added means that the linchpin will be absent from earlier states and present in later states, simulating the assertion of the linchpin. To-be-removed means that the linchpin will be present in earlier states but absent in later states, simulating the un-assertion of the linchpin. Each linchpin has a corresponding TPTP conjecture, from which we construct 2 TPTP problems, one before the elaboration and after. Hence the 2 TPTP problems have the same conjecture but different include files. For static linchpins, their conjectures should be answerable both before and after elaboration. For to-be-added linchpins, the conjectures should be unanswerable before elaboration and answerable after. For to-be-removed linchpins, the conjectures should be answerable before elaboration and unanswerable after.

After the linchpins and the conjectures were established, we attached "chaff" to the linchpins. This forms the *dynamic chaff* section of the KB mentioned earlier. The system finds synthetic terms and assertions that are not used in any justification of any TPTP problem, and marks them as potential chaff. Each linchpin is then assigned some chaff, with each linchpin getting approximately the same amount of chaff. The linchpin is inserted into the chaff, the list is randomized, and the resulting list of assertions is assigned to an include file.

To produce the order of the TPTP problems and elaborations, we randomly sorted the linchpins, and added two copies of its corresponding TPTP problem, once before the linchpin and once after. This forms the *TPTP transcript*, which is a series of operations -- either TPTP queries or linchpins (i.e. theory elaborations). The non-elaboration queries generated earlier were then added more or less randomly into the TPTP transcript. From this TPTP transcript we then produced actual TPTP problems with headers, correct lists of include files, and conjectures.

3.1.4 Query Set

During discussions with Dr. Geoff Sutcliffe, the organizer of all of TPTP, CASC and ESARLT, it became clear that our initial TPTP suite (hereafter referred to as the *Elaboration Suite*) was so far beyond what the current state of the art theorem provers can handle, that there was a significant risk that they would simply be ignored because they were too hard. In response to this, we designed a new, similar suite, called the *Scaling Suite*, comprising 6 sets of 50 problems, where each set increases the number of axioms in the background theory by an order of magnitude. Problems in the final set 6 are the same size as those in the Elaboration Suite, so the two suites maintain some overlap. The Scaling Suite encourages the community to tackle the suite as a whole because the smallest set 1 is already easily answerable by the current state of the art. The scaling suite will be released to the TPTP community this coming month, i.e., in September 2007.

At each of the 6 Problem Levels, the 50 problems comprised 5 examples each of 10 predicate types. The types selected were representative of those predicates with the highest utility, based on factors such as the total number uses. These types included:

- genls
  (genls Hammer Artifact): Is a hammer an artifact?

- isa
  (isa France Country): Is France a country?

- disjointWith
  (disjointWith Eagle Airplane): Is it true that no eagles are airplanes?

- value inheritance
  (relationAllInstance languageSpoken AustralianPerson EnglishLanguage):
  Do all Australians speak English?

- role filler existence
  (relationAllExists anatomicalParts Elephant Trunk-BodyPart)"
  Do all elephants have trunks?

- symmetry
  (bordersOn Germany France): Does German border on France (and vice versa)?

- transitivity
  (geographicalSubRegions Europe CityOfParisFrance): Is Paris, France in Europe?

- genlPreds
  (genlPreds mother relatives):
  If someone is a person's mother is she also their relative?

- Look ups (directly stated assertions)
  (capitalCity France CityOfParisFrance):
  Is Paris the capital of France?
  (i.e., Is this known to be a fact, without requiring any inference?)

3.2 FOL Theorem Prover Integration Feasibility Experiments

As a prelude to full integration as reasoning modules within the Cyc inference engine (which would have been expected in a fully-funded EP project), we carried out integration feasibility experiments using three leading theorem provers: Otter[5], E[6], and SPASS[7]. As already mentioned, we had previously discovered that none of these theorem provers is capable of loading the entire OpenCyc Knowledge Base as it currently stands. The design of the TPTP problem sets discussed above is closely tied to the results of our integration feasibility experiments. All of the theorem provers were capable of running the queries at both Level 1 (the minimal theory required to answer the particular question) and Level 2 (the minimal theory required to answer all of the questions). However, using either Level 1 or Level 2 as the basis of integrating the theorem provers into a confederated system is quite impractical, because it would require the dispatching system to know exactly which pieces of information would be necessary. In effect, the dispatching system would need to solve the problem before sending it out for one of the other systems to solve. Thus, experiments were focused on determining the systems' behavior at the higher levels (though not as high as Level 6, which is completely outside the realm of all of these systems).

Initial experiments with Otter showed that it is not capable of handling the larger problems that are of interest, apparently because of a hard-coded memory limit that causes it to halt early on all of our larger problems (anything more than a few thousand axioms).

Initial experiments with E showed that it works correctly on queries for which a proof exists, but runs infinitely for queries where no proof exists. Further experiments with E got around this problem by using an external timeout.

Initial SPASS experiments showed no hard limits on its utility.

3.2.1 KB Scaling Experiments

As an exploration of how SPASS scales with the number of irrelevant (chaff) axioms, we ran an experiment in which the TPTP problem generator was used to generate multiple TPTP problem sets with the same conjecture but different amounts of chaff. We

[5] See "*OTTER 3.3 Reference Manual*" by William McCune at http://www.cs.unm.edu/~mccune/otter/Otter33.pdf

[6] See "*E - a brainiac theorem prover,*" Journal of AI Communications 15(2):111-126

[7] See the "*SPASS: Combining superposition, Sorts and Splitting*" chapter in the "*Handbook of Automated Reasoning*" by C. Weidenbach, 1999, Elsevier.

generated a problem set of 12 queries, 8 of which were satisfiable and 4 of which were unsatisfiable. Of the satisfiable queries, 3 were simple lookups and 5 had multi-step proofs. These three categories of queries have different scaling properties, as seen in the following graph:

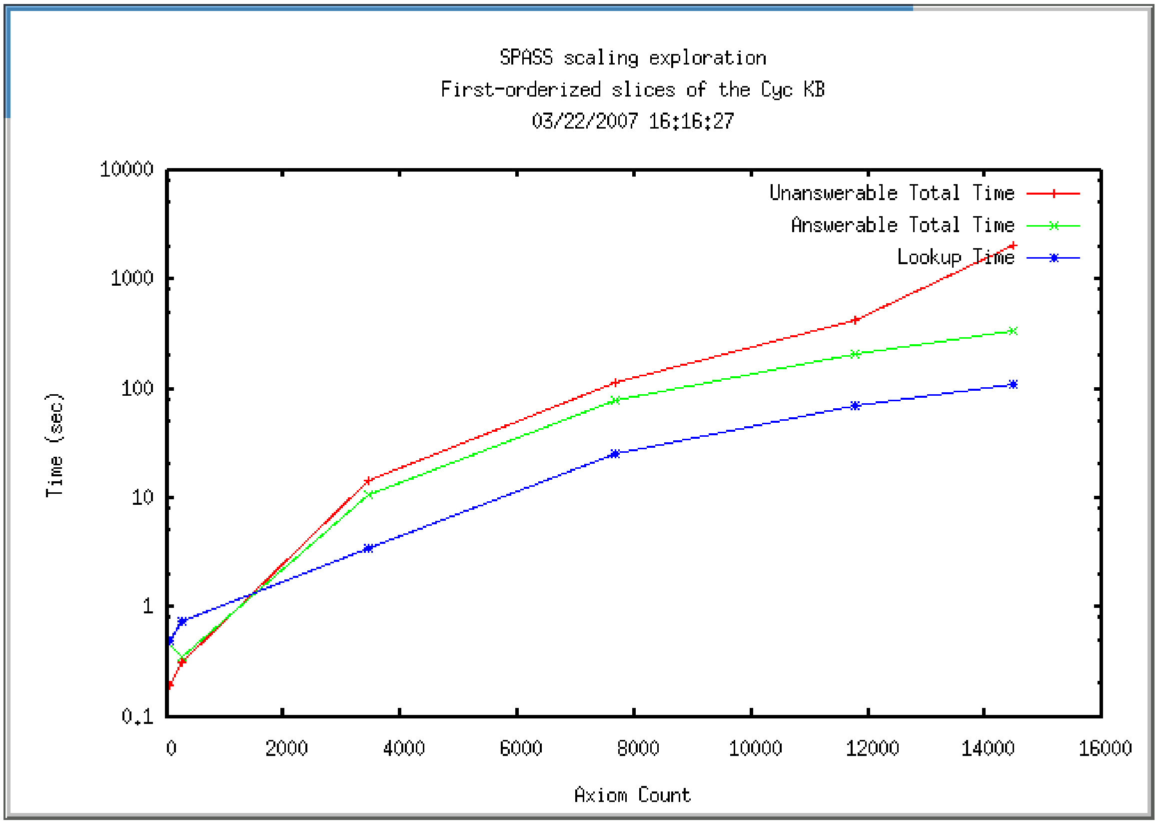


Figure 16. Time versus Theory Size with SPASS. The red line shows the amount of time it took to give up on unanswerable problems, the green line shows the time it took to answer problems requiring the chaining of multiple facts or rules, and the blue line shows how long it took to answer questions that involved simply looking up the answer in the data. For all question classes, <u>time to answer</u> appears to increase rapidly with the size of the theory.

This experiment shows that in its current incarnation, SPASS encounters substantial difficulty with larger theories. Even the lookup time appears to be increasing super-linearly and is already on the order of 100 seconds with just 14,000 axioms. As a reference point, Cyc contains millions of axioms, and can answer most lookup queries in milliseconds, or a small number of seconds if there is a page fault. We conclude that SPASS in its current form is not capable of efficiently handling large axiom sets, so if it is to perform reasonably on the Cyc TPTP challenge problem set, it will first need additional indexing support. If SPASS is representative of the state of the art among FOL theorem provers, many of them will need additional indexing support and probably more efficient KB data-structures in order to perform reasonably well on the Cyc TPTP challenge problem set.

An unfortunate consequence of this is that we could not directly test the Cyc TPTP challenge problem set for satisfiability. Instead, we were forced to make scaled-down forms of the challenge problem set, and then afterwards add additional chaff (i.e. the six levels of problems discussed earlier). Unless there is a bug in the TPTP problem generator, the chaff will not introduce a contradiction (which would allow intentionally unprovable problems to be proven). Given the slowness of FOL theorem provers (exemplified here by SPASS), we were not able to test conclusively for the lack of a contradiction, but could only deem it statistically unlikely by testing smaller versions of the problem set.

The results for very small numbers of axioms are interesting, but we refrain from drawing conclusions from them, since the variance in timing could well account for the data, especially with only 12 queries.

An interesting feature is found at the far right of the graph, where the unanswerable total time rises sharply (remember, the graph is on a log scale, so the increase is very sharp). Our hypothesis is that this is because of the connectedness properties of the synthetic KB and the way in which the TPTP problem generator chose the chaff to export. For this experiment, the TPTP problem generator selected a random subset of the chaff in the synthetic KB to export for each experiment. The rightmost data points are 100% of the synthetic KB that was generated for this set of experiments. We hypothesize that the KB connectedness increases sharply as the percentage of chaff exported approaches 100%, and that hence the theorem provers have more and deeper dead ends to attempt to prove. This hypothesis could be tested by experiments (not yet conducted) in which 100% of the synthetic KB is always exported, but in which we generate different synthetic KBs of different sizes. If our hypothesis is correct, we would expect that experiment to not show the sharp upturn in unanswerable total time, since all the chaff would have the same KB connectedness.

3.2.2 Expanded Transitivity Experiments

In April 2007, we expanded the set of tests to include queries involving:

- **Disjointness:**
  (disjointWith Dog Cat) → no individual cat is also a dog, and vice versa

- **Genlpreds:** (generalization over predicates)
  if (hotelInCity BerkshireHotel-London CityOfLondonEngland) is true,
  then the following are also true:
  - (hotelInRegion BerkshireHotel-London CityOfLondonEngland) *and*
  - (objectFoundInLocation BerkshireHotel-London CityOfLondonEngland) .

We also expanded our support for converting Cyc's KB into First-Order Logic by adding support for some additional transitivity:

- if (hotelInRegion BerkshireHotel-London CityOfLondonEngland) is true, then so are :
  - (hotelInRegion BerkshireHotel-London England) *and*
  - (hotelInRegion BerkshireHotel-London UnitedKingdomOfGreatBritainAndNorthernIreland).

These changes led to an interesting change in the behavior of SPASS (but not Otter or E), namely an enormous increase in the time taken to give up on unprovable problems. In previous experiments, the median time to give up on unprovable problems was only slightly longer than the median time to find a proof for provable problems. For example, in SPASS scaling exploration shown if 16 for a data point with a comparable number of axioms, the median answerable time was 0.58s and the median unanswerable time was 0.72s.

After the addition of the transitivity extensions to the problem set, the median answerable time was 0.62s and the median unanswerable time was 10118s (over 2.8 hours). Also interestingly, the time did not vary much depending on the predicate used in the conjecture. We hypothesize that the additional axioms we exported added enough connectivity (i.e., ensuring that the terms in the KBs we generated were connected to a similar degree as that in the original KB) and inferential fertility to allow SPASS to waste lots of time going down irrelevant paths.

3.2.3 Cyc versus Otter, SPASS, and E Performance on TPTP Problems

For an initial comparison of capabilities we investigated the theorem provers E and SPASS on disjointness queries in both proof-checker mode (where exactly and only the background theory axioms needed for the conjecture are provided) and in "elaboration mode" (which is the union of the background theories for proof-checker mode for the entire Elaboration suite of queries). The background theories for the proof-checker problems therefore had less than 100 axioms and the background theories for the elaboration mode problems had around 1600 axioms. The results are shown in Figure 17, below.

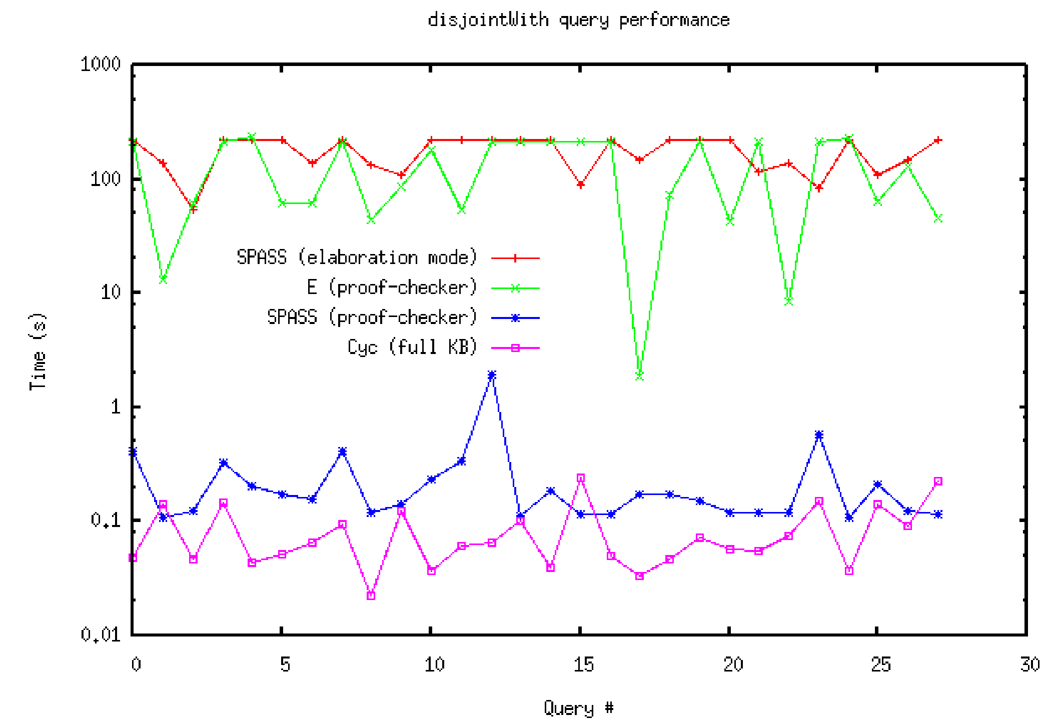


Figure 17. Cyc's performance on an example of TPTP problems versus various scaled versions of the problem for E and SPASS. Both E and SPASS were able to find solutions for the proof-checker mode experiments, although E was several orders of magnitude slower than Cyc, even with the greatly simplified problem space.

All queries were run with a 212 second timeout. For comparison, the performance of Cyc with its full background theory of 4 million axioms is also shown. SPASS performed 2-3 orders of magnitude faster than E in proof-checker mode, and was able to answer about half the queries in elaboration mode before timing out. E was only able to answer about half of the proof-checker queries before timing out, and was unable to answer any queries in elaboration mode before timing out.

Finally, note that Cyc performed several orders of magnitude faster than the theorem provers with a background theory that is many orders of magnitude larger than those given to the provers.

3.2.4 Relevant Chaff Experiment

It was conjectured that the process used by which chaff (i.e., axioms other than used needed to prove a conjecture) was selected might impact the resulting problem difficulty. To measure this, a set of experiments was conducted during the TPTP problem

generation that examined the effect of using potentially relevant chaff versus using random chaff. The problem difficulty was measured by the amount of time it took the theorem prover to find a proof (for provable queries), or the amount of time it took the theorem prover to give up looking for a proof (for unprovable queries). The overall results of these experiments were inconclusive; more detailed data is presented in Appendix B.

3.3 FOL community Results

Shortly after publishing the TPTP Scaling Suite and the initial version of the Elaboration Suite we ran Vampire, E, SPASS and Otter over the suites to determine their baseline performance. Their performance on the Scaling Suite is given below.

	Scaling Suite Number of problems answerable out of 50, time limit = 212 seconds			
	Vampire 9.0	**E 0.99 Singtom**	**SPASS v3.0c**	**Otter 3.3f**
Level 1	50	48	49	50
Level 2	50	27	37	10
Level 3	43	25	0	0
Level 4	13	10	0	0
Level 5	0	0	0	0
Level 6	0	0	0	0
Total	156 (52.0%)	110 (36.7%)	86 (28.7%)	60 (20.0%)

We also tested the initial version of the Elaboration Suite in proof checker mode. As expected, none of the theorem provers could currently answer any of the 300 problems.

We verified that the problems were actually answerable by generating proof-checker versions of each of the Elaboration Suite problems and evaluating performance on those.

	Elaboration Suite, proof-checker Number of problems answerable out of 50, time limit = 212 seconds			
	Vampire 9.0	**E 0.99 Singtom**	**SPASS v3.0c**	**Otter 3.3f**
Total	Not run	298 (99.3%)	293 (97.7%)	300 (100.0%)

4. Conclusions and Recommendations

4.1 Project Results

This project set out to explore ways to obtain the qualitative improvements in machine reasoning performance needed to ensure that the Intelligence Community will have a fighting chance of culling critical information from a rapidly increasing swarm of available data. Prior to this project, progress had been made on two somewhat orthogonal and independent fronts: 1) the automatic machine reasoning (aka theorem proving) community has been improving the efficiency with which deep reasoning can be performed within a fairly narrow theory and simple world model; and 2) Cyc has continued to expand its ability to reason over very rich and broad base of knowledge and address problems more akin to those faced by the IC, albeit trading increased representational fidelity and inferential complexity for speed. This project set out to identify opportunities to find synergies that leverage the strengths of both approaches.

Despite a severe down-scoping at the start of this project, significant progress was made towards these goals along two different dimensions. The first goal was to energy the theorem proving community to turn their attention to problems of increasing complexity by using Cyc to generate a set of scalable problems for them to tackle. Since the TPTP Problem Library is the main regression and benchmarking suite used by the automatic theorem proving community, providing a suite of more challenging problems in this syntax to via this channel facilitates the communities ability to begin addressing them.

Even if Cyc were to be removed from the picture at this point, these problem suites would provide fodder to that can bring the theorem proving technologies closer to addressing the types of problems critical to IC success. However, there is a second benefit to growing theorem proving technologies in these directions, namely an increased likelihood of Cyc being able to leverage some of these technologies to attack specific types of problems – or parts of problems – very efficiently. Since Cyc's architecture supports the assimilation of a panoply of problem solvers to attack any given problem, improvements in efficiency of one or more of these leading theorem provers is expected to have rapid payback in Cyc's ability to solve complex problems in realistic timeframes.

In addition to fostering advancements in the theorem proving community, this project also supported a series of experiments that enabled immediate and substantial improvements to the efficiency of Cyc's current reasoning capabilities. A variety of techniques were examined and implemented, including the use of reinforcement learning, reuse of problem-solving knowledge from past problems to address later similar ones, and the clustering of knowledge used to solve specific problem types, thereby greatly reducing the overall search space. While these experiments yielded significant results, they also served to identify likely paths forward for even more significant improvements.

4.2 The Path Forward: Further Enhancing Cyc's Reasoning

As described above, the experiments with Cyc's current inference capabilities led to some immediate improvements in Cyc's reasoning efficiency as well as to a long list of additional opportunities. The most promising of these are outlined here:

4.2.1 Proof Spec Tactician

- **Make Strategist automatically use Proof Spec Tactician**

  Currently the Strategist only uses the Proof Spec Tactician when an explicit proof spec is passed in via an inference parameter. The Strategist could be extended to automatically extract relevant proof specs for a query from Cyc's case store of existing proof specs. If resources permit, it can then fall back to general discovery proof after first attempting the relevant proof specs.

- **Store and access sub proof specs as well**

  Currently Cyc only stores top-level proof specs that applied to an entire inference. We can investigate extending that to sub proof-specs as well, which would enable more cross-query reuse between queries that are not analogous at the top-level but share common subqueries that are analogous.

- **Work offline to expand proof store**

  The proof store is built from experience in general discovery proof. Therefore, we can leverage an arbitrary amount of offline inference to increase the proof store. This could be achieved by asking queries analogous to those already in the store.

- **Explore alternate reasoners for case generalization and abstraction**

  The efficient storage and retrieval of relevant proof stores is itself an interesting problem to investigate and optimize. Even with a perfect Proof Spec Tactician we still need to be able to access a relevant proof spec more quickly than the savings it provides in order to be a net performance gain. This can be viewed as a form of analogy reasoning so existing analogy reasoners could be evaluated here.

- **Profiling sweep of propagation code**

  Effort on the current Proof Spec Tactician was primarily focused on correctness rather than efficiency. We should perform a profiling analysis on the Proof Spec Tactician and investigate reducing the expense of the bookkeeping overhead on the propagation of proof-specs.

4.2.2 TPTP

- **Formally publish Elaboration Suite**

  Currently, only the Scaling Suite has been published to the TPTP community. We should publish the Elaboration Suite as well.

- **Update both suites each year**

  Both the Scaling and Elaboration Suites should be updated with newly generated problems at least yearly.

- **Extend from OpenCyc to ResearchCyc**

  Currently, only OpenCyc KB Content is used to generate the TPTP suites. We should extend that to use ResearchCyc so that ATP developers that have a ResearchCyc license can work on a version that even more closely mirrors ResearchCyc content and behavior.

- **Extend reasoning modules covered**

  We should explore extending the TPTP generation modules to cover more cases of Cyc subtheory reasoning.

- **Improve relevant chaff generation algorithm**

  Our current relevant chaff generation algorithm appears to be indistinguishable in effect from just random selection of chaff to include. We should investigate whether this is due to a flaw in our approach and whether paying closer attention to axiom connectivity would behave better.

4.2.3 Rule Connectivity Graph

- **Extend to rule sets**

  Currently the rule connectivity heuristic is based on pairwise rule success. We should investigate whether or not extending this to remember actual sets of successful rules rather than just pairwise success would maintain generality and improve focus without overfitting.

- **Learn heuristic weighting**

  We should perform a hill-climbing walk through heuristic weight space to learn the best a priori weighting for the rule connectivity heuristic in relation to the other existing inference heuristics.

4.2.4 Reinforcement Learning Tactician

- **Implement join iteratively**

  When working on a join of A(x) ^ B(x) the tactician should interleave iterative work on A(x) and B(x) so that we don't have to entirely finish one problem (e.g. A(x) or B(x)) before we start working on the other problem (e.g. B(x) or A(x)).

- **Reward answer count**

  The Reinforcement Learning Tactician learned a policy that favored time to first answer by sacrificing time to later answers. We should device a policy that fairly factors in later answers and repeat the learning experiment under the new policy.

4.3 The Path Forward: Harnessing the Machine Reasoning Community

The final conclusion is that the overall hypothesis appears to be empirically verified, to the extent we have been able to do these experiments: a group of *n* of the world's leading theorem-provers "harnessed together" ought to be able to solve many of the problems, especially the IC-sized Level 6 problems, more than *n* times faster than any one of them (any one of the theorem proving programs) could, working on that same problem in its entirety all by itself.

The path forward from this point includes continuing to interact with, engage, and challenge the worldwide ATP community, to get them as much as possible to adopt this challenge as their own. Without participation by Cyc or other U.S. Government funded researchers and developers, and without some of the problems coming from (or motivated by) IARPA or the IC more broadly, the ATP community is likely to once again become diverted toward fine-tuning against the existing problem set rather than coming up with new, IC-sized and IC-relevant challenge problems.

We hope that this resource – the now-engaged worldwide ATP community – is not left "on the shelf." We strongly urge IARPA to follow through with the original plan to harness that resource, even if Cyc is only one of many players in that endeavor.

5. References

1. Kaebling,Leslie, http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html.

2. Wikipedia Reinforcement Learning Overview, http://en.wikipedia.org/wiki/Reinforcement_learning

3. Oblinger, D. 2005. Darpa Transfer Learning Program. http://www.darpa.mil/ipto/solicitations/closed/05-29 PIP.htm .

4. Ramachandran, Deepak, Reagan P., and Goolsbey, K., First-Orderized ResearchCyc: Expressivity and Efficiency in a Common-Sense Ontology, *Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications.* Pittsburgh, Pennsylvania, July 2005.

5. McCune, William, "*OTTER 3.3 Reference Manual*" http://www.cs.unm.edu/~mccune/otter/Otter33.pdf

6. *"E - a brainiac theorem prover,"* Journal of AI Communications 15(2):111-126

7. Weidenbach, C., "*SPASS: Combining superposition, Sorts and Splitting*" chapter in the "*Handbook of Automated Reasoning*", 1999.

6. List of Symbols, Abbreviations and Acronyms

ATP – Automatic Theorem Proving

Cyc – The Cyc Knowledge Base

FOL – First Order Logic

HL – Heuristic Level

IC – Intelligence Community

KB – Knowledge Base

RL – Reinforcement Learning

TPTP – Thousands of Problems for Theorem Provers

Appendix A. CADE-21/ESARLT Talk Overview

The following is an overview of an invited talk entitled ***Cyc Design Challenges and Solutions*** that was presented by Keith Goolsbey at the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories (ESARLT) on July 15th in Bremen, Germany.

Abstract

> Cyc comprises a large, contextualized, common sense knowledge base (KB) which is encoded in an expressive representation language (essentially FOL with a few key extensions) and paired with an inference engine optimized for the classes of queries we most frequently encounter. These queries tend to mix relatively shallow reasoning within one of a large number of idiosyncratic subtheories with relatively deep reasoning within one of a very small number of stylized subtheories. The constraints of these queries in a large and expressive KB combined with the need to efficiently react to KB elaboration together provide a unique set of design challenges that are extremely stressful for the solutions provided by the current state of the art FOL theorem provers. The solutions to these challenges currently adopted by the Cyc inference engine will be presented within the context of a new suite of TPTP problems that are derived from Cyc's KB and typical queries and are intended to demonstrate Cyc's design challenges for investigation by the wider community.

Background on Cyc and CycL

CycL is essentially a first-order logic (FOL) with a few useful higher-order logic (HOL) extensions. The Cyc knowledge base (KB) is a very large common-sense knowledge base with millions of assertions over hundreds of thousands of terms. CycL has two components, the Epistemological Language (EL), which is the external language of expression and semantics, and the Heuristic Language (HL), which is the internal language of implementation. We translate between the EL and the HL to have both an expressive language for representation and a powerful language for reasoning.

OpenCyc based TPTP Suite

Cycorp announced that it is releasing to the CADE community a new TPTP suite of 300 problems based on OpenCyc KB content and queries. The suite is derived from OpenCyc and as such is freely available to anyone to use for any purpose. It is Cycorp’s hope that by extending TPTP to include queries that express the design constraints of Cyc the CADE community will begin to address these constraints and therefore make their systems more directly useful for Cyc inference.

FOLification

FOLification is the process of converting CycL into a FOL for use in TPTP problems on FOL theorem provers in the CADE community. An extremely high percentage of OpenCyc is FOLifiable (> 89%).

Large, Persistent KB

The time spent to load a background theory cannot be ignored when the background theory is millions of axioms. It is best to reuse the background theory for multiple queries by maintaining state for the background theory between queries and to separate the store of background theory clauses from the store of working clauses in inference. Where possible, the background theory should be swapped in and out of memory lazily from disk via indexing since a given query only uses a small portion of it at any one time.

Indexing

Although the KB is large, comprehensive indexing allows for focused access to subsets of the KB that match expected unification patterns. This allows for tractable working sets during inference even with an immense KB. The SPASS theorem prover is distracted by irrelevant chaff which is an indication that better indexing is warranted.

Elaboration Tolerance

John McCarthy defined Elaboration Tolerance as follows:

> A formalism is *elaboration tolerant* to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances. Representations of information in natural language have good elaboration tolerance when used with human background knowledge. Human-level AI will require representations with much more elaboration tolerance than those used by present AI programs, because human-level AI needs to be able to take new phenomena into account.

The Cyc KB must be elaboration tolerant because we constantly do theory revision and need to query the KB quickly after each revision. This limits optimizations that perform extensive global analysis or assume a read-only knowledge base.

We simulate elaboration tolerance in the TPTP suite via the inclusion or omission of include files. Inclusion of an include file mimics either the creation of a term or asserting something. Exclusion of an include file mimics unasserting something or removing a term.

Pragmatic Incompleteness

Pragmatic incompleteness is just as bad as theoretical incompleteness in practice. When dealing with very large theories, the community needs to accept incompleteness and instead focus on metrics over a representative corpus of queries, such as time to first answer, time per answer and answerability within a given amount of time.

Optimized Subtheories

Subthoeries of interest are worth providing optimized HL representations and inference modules for the reasoner to use. The CADE community already does this with equality reasoning (for example, the paramodulation inference rule). Subtheories of interest to commonsense reasoning in general and Cyc in particular include: genls, isa, disjointness, value inheritance, role filler existence, symmetry, transitivity, predicate generalization and predicate negation. The performance of Cyc on disjointness queries is compared to SPASS, E and Otter which all currently lack optimized support for a disjointness subtheory.

Microtheories

A lattice of microtheories in Cyc tersely supports a large number of theories via inheritance via the genlMt predicate. Cyc dynamically reasons about microtheory subsumption and relevance in the tightest inner loop of inference at runtime. This has an impact on FOLification and usually confounds the unit propagation heuristic used by FOL provers.

Meta-Reasoning

Cyc manages the complexity of its inference rules (called the Worker) via explicit meta reasoning (called the Tactician) and meta-meta reasoning (called the Strategist). The desirable division of computational labor is usually about 90% Worker, 9% Tactician and 1% Strategist. We demonstrated the power of meta-meta reasoning by showing results of an experiment where meta-meta reasoning substantially sped up trivial queries.

Presentation and Reception

The talk was well-received and the CADE community appears to be interested in tackling Cyc-scale problems. During each CADE conference the CADE ATP System Competition (CASC) is a formal competition held between theorem provers of the CADE community over a set of problems selected from the full TPTP suite.

The next CASC competition will be the first to include TPTP from the Cyc TPTP suite! There will be a new batch division with very large theories and there will be a Cyc subdivision. Cycorp will sponsor a monetary prize to the prover that does the best on the Cyc subdivision at the next CASC competition.

Scaling Suite vs. Elaboration Suite

During discussions with Dr. Geoff Sutcliffe, the organizer of all of TPTP, CASC and ESARLT, it became clear that our initial TPTP suite (we'll refer to this from now on as the *Elaboration Suite*) is so beyond what the current state of the art can handle that we run the risk of being ignored for being too hard. Instead, we designed a new, similar suite which we will call the *Scaling Suite*. This new suite will be 6 sets of 50 problems, where each set increases the number of axioms in the background theory by an order of magnitude. Problems in the final set 6 will be the same size as those in the Elaboration Suite, so the two suites will maintain some overlap. The Scaling Suite encourages the community to tackle the suite as a whole because the smallest set 1 is already easily answerable by the current state of the art. Work by Cycorp on developing and releasing the Scaling Suite will proceed in August for release and inclusion in the next planned TPTP release, currently slated for September 2007.

Appendix B. Relevant Chaff Experiments

During development of the Scaling Suite we ran an experiment with our TPTP problem generation we called the Relevant Chaff Experiment, which is described below.

In the TPTP problem set, a small number of axioms, usually between 1 and 40, are used to prove the conjecture. These are the "wheat" and all the other axioms are the "chaff". This experiment tested the hypothesis that using potentially relevant chaff would create more difficult problems than choosing random chaff. We measured difficulty by the amount of time it took the theorem prover to find a proof (for provable queries), or the amount of time it took the theorem prover to give up looking for a proof (for unprovable queries).

We constructed two sibling problem sets with the same conjectures -- 10 provable disjointWith queries and 10 unprovable disjointWith queries. In one problem set, we used an algorithm to select potentially relevant chaff axioms to add to the "wheat". In the other, we randomly selected chaff axioms to add to the "wheat". We called the first problem set the "potentially relevant chaff problem set" and the second the "irrelevant chaff problem set". Each of these two problem sets contained exactly 200 axioms and 1 conjecture.

The algorithm used to generate the relevant chaff is roughly as follows:

Iterate over the "wheat" axioms. For each "wheat" axiom W,

- Select a chaff generation module M from the following probability distribution:

chaff-generator-transitive-generality-estimate	.45
chaff-generator-random-gaf-for-term	.40
chaff-generator-random-gaf-with-same-predicate	.10
chaff-generator-random-gaf-with-spec-predicate	.05

- Apply module M to axiom W. If this yields an axiom, add it to the set of chaff axioms.

The chaff generation module algorithms are as follows, assuming an input axiom W:

- **chaff-generator-transitive-generality-estimate**
  Applies only to axioms with transitive predicates. Select an axiom sharing an argument with W and with the other argument near the top of the ontology.

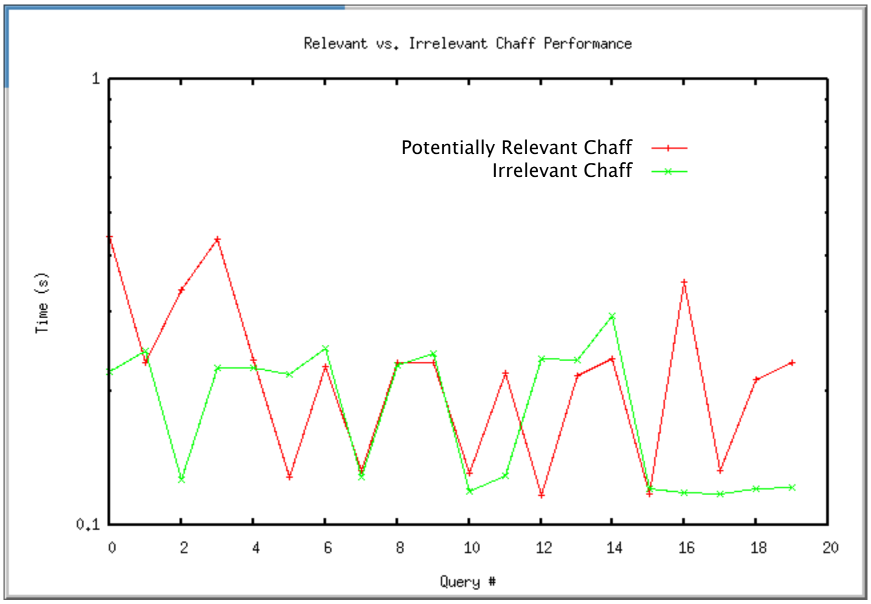
- **chaff-generator-random-gaf-for-term**
  Select a random gaf that shares a predicate and a term in the arg1 or arg2 position with W

- **chaff-generator-random-gaf-with-same-predicate**
  Select a random gaf with the same predicate as W

- **chaff-generator-random-gaf-with-spec-predicate**
  Select a random gaf with some specialization of W's predicate

We ran this experiment on three ATPs: Otter, SPASS, and E. The experiments were run concurrently on a 4-CPU machine with 8 GB of RAM, where each CPU is a dual-core AMD Opteron. The results, shown over the next three graphs, were inconclusive:

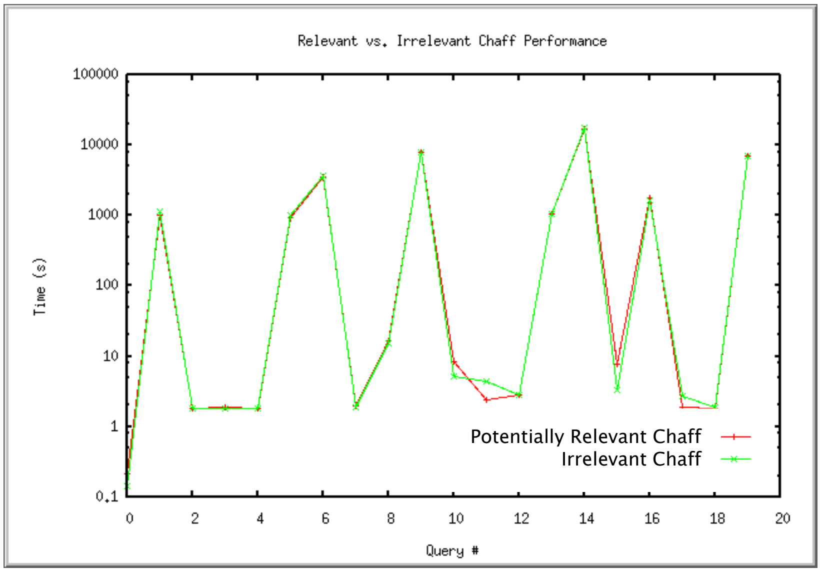
Otter



Statistical Analysis (potentially relevant chaff)	Statistical Analysis (irrelevant chaff)
N : 20	N : 20
Min : 0.1165s	Min : 0.1173s
Max : 0.4439s	Max : 0.2944s
Median : 0.2284s	Median : 0.219s
Mean : 0.2298s	Mean : 0.1858s
Standard Deviation : 0.0951	Standard Deviation : 0.05956

The data from Otter weakly supports our hypothesis that potentially relevant chaff is more distracting than random chaff, but more data would be needed to prove anything conclusively.

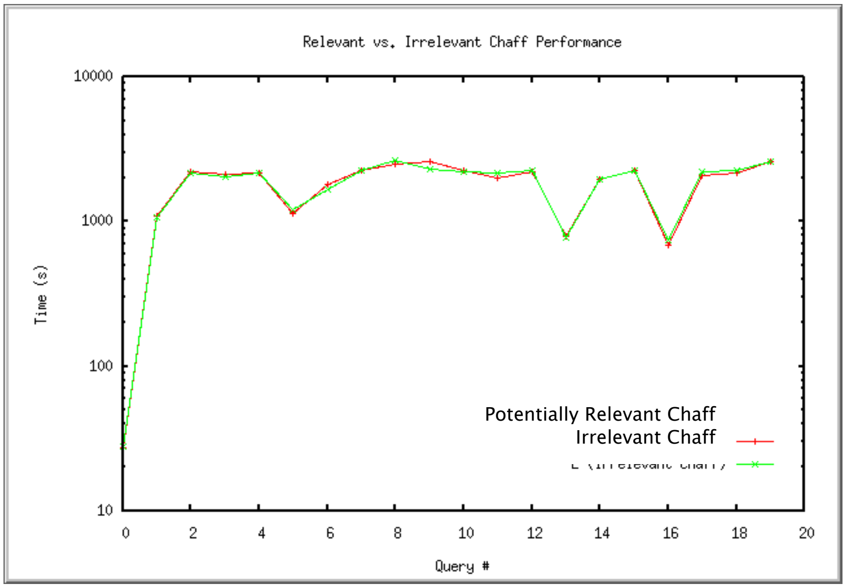
SPASS



Statistical Analysis (potentially relevant chaff)	Statistical Analysis (irrelevant chaff)
N : 20	N : 20
Min : 0.2155	Min : 0.1415
Max : 17350	Max : 18070
Median : 8.074	Median : 4.786
Mean : 2036	Mean : 2079
Standard Deviation : 4181	Standard Deviation : 4311

The data from SPASS also weakly supports our hypothesis that potentially relevant chaff is more problematic than irrelevant chaff, but more data would be needed to prove anything conclusively.

E



Statistical Analysis (potentially relevant chaff)	Statistical Analysis (irrelevant chaff)
N : 20	N : 20
Min : 27.24	Min : 27.92
Max : 2617	Max : 2624
Median : 2134	Median : 2156
Mean : 1843	Mean : 1841
Standard Deviation : 690.8	Standard Deviation : 682.7

The data from E differs so little between the two conditions that we can't conclude anything useful from it, except that the relevance of chaff doesn't apart to be particularly significant in this implementation of E.

We suspect that the effects of the relevant versus irrelevant chaff may be being drowned out due to the lack of indexing support in ATPs. If any of these ATPs use a clause tree internal representation, potentially relevant chaff would be represented more efficiently, so this might offset the indexing-related slowdown. The fact that the number of facts that are potentially relevant to the problem seems to have no effect on the time to answer the problem was quite surprising to us, as we expected the systems to spend more time

solving problems when the number of easily accessible inference paths was higher. The degree to which any of these systems are affected by the amount of potentially relevant chaff appears to be un-testable until the ATPs have improved their indexing support.

We wish to thank Geoff Sutcliffe, Andrei Voronkov, and Michael Genesereth, for their substantive technical collaboration, and Eric Rickard, DTO, and AFRL for their support of this work, but all the opinions expressed herein reflect those of the authors, not necessarily those of the United States Government nor of those individuals.

Questions, suggestions, and other feedback are welcome, and should be directed to the PI of this project: Doug Lenat doug@cyc.com .